Package 'spOccupancy'

November 25, 2021

Type Package

Title Single Species, Multispecies, and Integrated Spatial Occupancy Models

Version 0.1.3

Author Jeffrey Doser [aut, cre],
Andrew Finley [aut]

Maintainer Jeffrey Doser <doserjef@msu.edu>

Description Fits single species, multispecies, and integrated non-spatial and spatial occupancy models using Markov Chain Monte Carlo (MCMC). Models are fit using Polya-Gamma data augmentation detailed in Polson, Scott, and Windle (2013) <doi:10.1080/01621459.2013.829001>. Spatial models are fit using either Gaussian processes or Nearest Neighbor Gaussian Processes (NNGP) for large spatial datasets. Details on NNGP models are given in Datta, Banerjee, Finley, and Gelfand (2016) <doi:10.1080/01621459.2015.1044091> and Finley, Datta, and Banerjee (2020) <arXiv:2001.09111>. Provides functionality for data integration of multiple single species occupancy data sets using a joint likelihood framework. Details on data integration are given in Miller, Pacifici, Sanderlin, and Reich (2019) <doi:10.1111/2041-210X.13110>. Details on single species and multispecies models are found in MacKenzie, Nichols, Lachman, Droege, Royle, and Langtimm (2002) <doi:10.1890/0012-9658(2002)083[2248:ESORWD]2.0.CO;2> and Dorazio and Royle <doi:10.1198/016214505000000015>, respectively.

License GPL (>= 3)

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

URL https://www.jeffdoser.com/files/spoccupancy-web, https://github.com/doserjef/spOccupancy

BugReports https://github.com/doserjef/spOccupancy/issues

Imports stats, coda, abind, RANN, lme4, foreach, doParallel, spBayes

NeedsCompilation yes

Repository CRAN

Date/Publication 2021-11-25 06:30:02 UTC

R topics documented:

`fitted.intPGOcc` *Extract Model Fitted Values for intPGOcc Object*

Description

Method for extracting model fitted values from a fitted single species integrated occupancy (`intPGOcc`) model.

Usage

```
## S3 method for class 'intPGOcc'
fitted(object, ...)
```

Arguments

`object`	object of class `intPGOcc`.
`...`	currently no additional arguments

Details

A method to the generic `fitted` function to extract fitted values for fitted model objects of class `intPGOcc`.

Value

A list of three-dimensional numeric arrays of fitted values for each individual data source for use in Goodness of Fit assessments.

`fitted.msPGOcc` *Extract Model Fitted Values for msPGOcc Object*

Description

Method for extracting model fitted values from a fitted multispecies occupancy (`msPGOcc`) model.

Usage

```
## S3 method for class 'msPGOcc'
fitted(object, ...)
```

Arguments

`object`	object of class `msPGOcc`.
`...`	currently no additional arguments

Details

A method to the generic `fitted` function to extract fitted values for fitted model objects of class `msPGOcc`.

Value

A four-dimensional numeric array of fitted values for use in Goodness of Fit assessments. Array dimensions correspond to MCMC samples, species, sites, and replicates.

fitted.PGOcc *Extract Model Fitted Values for PGOcc Object*

Description

Method for extracting model fitted values from a fitted single species occupancy (PGOcc) model.

Usage

```
## S3 method for class 'PGOcc'
fitted(object, ...)
```

Arguments

object	object of class PGOcc.
...	currently no additional arguments

Details

A method to the generic fitted function to extract fitted values for fitted model objects of class PGOcc.

Value

A three-dimensional numeric array of fitted values for use in Goodness of Fit assessments. Array dimensions correspond to MCMC samples, sites, and replicates.

fitted.spIntPGOcc *Extract Model Fitted Values for spIntPGOcc Object*

Description

Method for extracting model fitted values from a fitted single species integrated spatial occupancy (spIntPGOcc) model.

Usage

```
## S3 method for class 'spIntPGOcc'
fitted(object, ...)
```

Arguments

object	object of class spIntPGOcc.
...	currently no additional arguments

Details

A method to the generic `fitted` function to extract fitted values for fitted model objects of class `spIntPGOcc`.

Value

A list of three-dimensional numeric arrays of fitted values for each individual data source for use in Goodness of Fit assessments.

`fitted.spMsPGOcc` *Extract Model Fitted Values for spMsPGOcc Object*

Description

Method for extracting model fitted values from a fitted multispecies spatial occupancy (`spMsPGOcc`) model.

Usage

```
## S3 method for class 'spMsPGOcc'
fitted(object, ...)
```

Arguments

`object`	object of class `spMsPGOcc`.
`...`	currently no additional arguments

Details

A method to the generic `fitted` function to extract fitted values for fitted model objects of class `spMsPGOcc`.

Value

A four-dimensional numeric array of fitted values for use in Goodness of Fit assessments. Array dimensions correspond to MCMC samples, species, sites, and replicates.

fitted.spPGOcc *Extract Model Fitted Values for spPGOcc Object*

Description

Method for extracting model fitted values from a fitted single species spatial occupancy (spPGOcc) model.

Usage

```
## S3 method for class 'spPGOcc'
fitted(object, ...)
```

Arguments

object	object of class spPGOcc.
...	currently no additional arguments

Details

A method to the generic fitted function to extract fitted values for fitted model objects of class spPGOcc.

Value

A three-dimensional numeric array of fitted values for use in Goodness of Fit assessments. Array dimensions correspond to MCMC samples, sites, and replicates.

hbef2015 *Detection-nondetection data of 12 foliage gleaning bird species in 2015 in the Hubbard Brook Experimental Forest*

Description

Detection-nondetection data of 12 foliage gleaning bird species in 2015 in the Hubbard Brook Experimental Forest (HBEF) in New Hampshire, USA. Data were collected at 373 sites over three replicate point counts each of 10 minutes in length, with a detection radius of 100m. Some sites were not visited for all three replicates. The 12 species included in the data set are as follows: (1) AMRE: American Redstart; (2) BAWW: Black-and-white Warbler; (3) BHVI: Blue-headed Vireo; (4) BLBW: Blackburnian Warbler; (5) BLPW: Blackpoll Warbler; (6) BTBW: Black-throated Blue Warbler; (7) BTNW: BLack-throated Green Warbler; (8) CAWA: Canada Warbler; (9) MAWA: Magnolia Warbler; (10) NAWA: Nashville Warbler; (11) OVEN: Ovenbird; (12) REVI: Red-eyed Vireo.

Usage

```
data(hbef2015)
```

Format

`hbef2015` is a list with four elements:

`y`: a three-dimensional array of detection-nondetection data with dimensions of species (12), sites (373) and replicates (3).

`occ.covs`: a numeric matrix with 373 rows and one column consisting of the elevation at each site.

`det.covs`: a list of two numeric matrices with 373 rows and 3 columns. The first element is the day of year when the survey was conducted for a given site and replicate. The second element is the time of day when the survey was conducted.

`coords`: a numeric matrix with 373 rows and two columns containing the site coordinates (Easting and Northing) in UTM Zone 19. The proj4string is "+proj=utm +zone=19 +units=m +datum=NAD83".

Source

Rodenhouse, N. and S. Sillett. 2019. Valleywide Bird Survey, Hubbard Brook Experimental Forest, 1999-2016 (ongoing) ver 3. Environmental Data Initiative. doi: 10.6073/pasta/faca2b2cf2db9d415c39b695cc7fc217 (Accessed 2021-09-07)

References

Doser, J.W., Leuenberger, W., Sillett, T.S., Hallworth, M.T., Zipkin, E.F. Integrated community occupancy models: A framework to assess occurrence and biodiversity dynamics using multiple data sources. https://arxiv.org/abs/2109.01894

---

hbefElev *Elevation in meters extracted at a 30m resolution across the Hubbard Brook Experimental Forest*

Description

Elevation in meters extracted at a 30m resolution of the Hubbard Brook Experimental Forest. Data come from the National Elevation Dataset.

Usage

```
data(hbefElev)
```

Format

hbefElev is a data frame with three columns:

val: the elevation value in meters.

Easting: the x coordinate of the point. The proj4string is "+proj=utm +zone=19 +units=m +datum=NAD83".

Northing: the y coordinate of the point. The proj4string is "+proj=utm +zone=19 +units=m +datum=NAD83".

Source

Gesch, D., Oimoen, M., Greenlee, S., Nelson, C., Steuck, M., & Tyler, D. (2002). The national elevation dataset. Photogrammetric engineering and remote sensing, 68(1), 5-32.

References

Gesch, D., Oimoen, M., Greenlee, S., Nelson, C., Steuck, M., & Tyler, D. (2002). The national elevation dataset. Photogrammetric engineering and remote sensing, 68(1), 5-32.

---

intPGOcc *Function for Fitting Single Species Integrated Occupancy Models Using Polya-Gamma Latent Variables*

Description

Function for fitting single species integrated occupancy models using Polya-Gamma latent variables. Data integration is done using a joint likelihood framework, assuming distinct detection models for each data source that are each conditional on a single latent occurrence process.

Usage

```
intPGOcc(occ.formula, det.formula, data, inits, priors, n.samples,
         n.omp.threads = 1, verbose = TRUE, n.report = 1000,
         n.burn = round(.10 * n.samples), n.thin = 1,
         k.fold, k.fold.threads = 1, k.fold.seed,
         k.fold.data, ...
)
```

Arguments

occ.formula	a symbolic description of the model to be fit for the occurrence portion of the model using R's model syntax. Only right-hand side of formula is specified. See example below.
det.formula	a list of symbolic descriptions of the models to be fit for the detection portion of the model using R's model syntax for each data set. Each element in the list is a formula for the detection model of a given data set. Only right-hand side of formula is specified. See example below.

`data` a list containing data necessary for model fitting. Valid tags are `y`, `occ.covs`, `det.covs`, and `sites`. `y` is a list of matrices or data frames for each data set used in the integrated model. Each element of the list has first dimension equal to the number of sites with that data source and second dimension equal to the maximum number of replicates at a given site. `occ.covs` is a matrix or data frame containing the variables used in the occupancy portion of the model, with the number of rows being the number of sites with at least one data source for each column (variable). `det.covs` is a list of variables included in the detection portion of the model for each data source. `det.covs` should have the same number of elements as `y`, where each element is itself a list. Each element of the list for a given data source is a different detection covariate, which can be site-level or observational-level. Site-level covariates are specified as a vector with length equal to the number of observed sites of that data source, while observation-level covariates are specified as a matrix or data frame with the number of rows equal to the number of observed sites of that data source and number of columns equal to the maximum number of replicates at a given site.

`inits` a list with each tag corresponding to a parameter name. Valid tags are `z`, `beta`, and `alpha`. The value portion of tags `z` and `beta` is the parameter's initial value. The tag `alpha` is a list comprised of the initial values for the detection parameters for each data source. Each element of the list should be a vector of initial values for all detection parameters in the given data source or a single value for each data source to assign all parameters for a given data source the same initial value. See `priors` description for definition of each parameter name.

`priors` a list with each tag corresponding to a parameter name. Valid tags are `beta.normal` and `alpha.normal`. Occurrence (`beta`) and detection (`alpha`) regression coefficients are assumed to follow a normal distribution. For `beta` hyperparameters of the normal distribution are passed as a list of length two with the first and second elements corresponding to the mean and variance of the normal distribution, which are each specified as vectors of length equal to the number of coefficients to be estimated or of length one if priors are the same for all coefficients. For the detection coefficients `alpha`, the mean and variance hyperparameters are themselves passed in as lists, with each element of the list corresponding to the specific hyperparameters for the detection parameters in a given data source. If not specified, prior means are set to 0 and prior variances set to 2.72.

`n.samples` the number of posterior samples to collect.

`n.omp.threads` a positive integer indicating the number of threads to use for SMP parallel processing. The package must be compiled for OpenMP support. For most Intel-based machines, we recommend setting `n.omp.threads` up to the number of hypterthreaded cores. Note, `n.omp.threads` > 1 might not work on some systems.

`verbose` if `TRUE`, messages about data preparation, model specification, and progress of the sampler are printed to the screen. Otherwise, no messages are printed.

`n.report` the interval to report MCMC progress.

`n.burn` the number of samples out of the total `n.samples` to discard as burn-in. By default, the first 10% of samples is discarded.

`n.thin` the thinning interval for collection of MCMC samples. The thinning occurs after the `n.burn` samples are discarded. Default value is set to 1.

k.fold	specifies the number of k folds for cross-validation. If not specified as an argument, then cross-validation is not performed and k.fold.threads and k.fold.seed are ignored. In k-fold cross-validation, the data specified in data is randomly partitioned into k equal sized subsamples. Of the k subsamples, k - 1 subsamples are used to fit the model and the remaining k samples are used for prediction. The cross-validation process is repeated k times (the folds). As a scoring rule, we use the model deviance as described in Hooten and Hobbs (2015). Cross-validation is performed after the full model is fit using all the data. Cross-validation results are reported in the k.fold.deviance object in the return list.
k.fold.threads	number of threads to use for cross-validation. If k.fold.threads > 1 parallel processing is accomplished using the **foreach** and **doParallel** packages. Ignored if k.fold is not specified.
k.fold.seed	seed used to split data set into k.fold parts for k-fold cross-validation. Ignored if k.fold is not specified.
k.fold.data	an integer specifying the specific data set to hold out values from. If not specified, data from all data set locations will be incorporated into the k-fold cross-validation.
...	currently no additional arguments

Value

An object of class intPGOcc that is a list comprised of:

beta.samples	a coda object of posterior samples for the occupancy regression coefficients.
alpha.samples	a coda object of posterior samples for the detection regression coefficients for all data sources.
z.samples	a coda object of posterior samples for the latent occupancy values
psi.samples	a coda object of posterior samples for the latent occupancy probability values
y.rep.samples	a list of three-dimensional arrays of fitted values for each data source for use in Goodness of Fit assessments.
run.time	execution time reported using proc.time().
k.fold.deviance	scoring rule (deviance) from k-fold cross-validation. A separate deviance value is returned for each data source. Only included if k.fold is specified in function call. Only a single value is returned if k.fold.data is specified.

The return object will include additional objects used for subsequent prediction and/or model fit evaluation.

Note

Some of the underlying code used for generating random numbers from the Polya-Gamma distribution is taken from the **pgdraw** package written by Daniel F. Schmidt and Enes Makalic. Their code implements Algorithm 6 in PhD thesis of Jesse Bennett Windle (2013) https://repositories.lib.utexas.edu/handle/2152/21842.

Author(s)

Jeffrey W. Doser <doserjef@msu.edu>,
Andrew O. Finley <finleya@msu.edu>

References

Polson, N.G., J.G. Scott, and J. Windle. (2013) Bayesian Inference for Logistic Models Using Polya-Gamma Latent Variables. *Journal of the American Statistical Association*, 108:1339-1349.

Hooten, M. B., and Hobbs, N. T. (2015). A guide to Bayesian model selection for ecologists. Ecological monographs, 85(1), 3-28.

Finley, A. O., Datta, A., and Banerjee, S. (2020). spNNGP R package for nearest neighbor Gaussian process models. arXiv preprint arXiv:2001.09111.

Examples

```
set.seed(1008)

# Simulate Data ------------------------------------------------------------
J.x <- 15
J.y <- 15
J.all <- J.x * J.y
# Number of data sources.
n.data <- 4
# Sites for each data source.
J.obs <- sample(ceiling(0.2 * J.all):ceiling(0.5 * J.all), n.data, replace = TRUE)
# Replicates for each data source.
n.rep <- list()
for (i in 1:n.data) {
  n.rep[[i]] <- sample(1:4, size = J.obs[i], replace = TRUE)
}
# Occupancy covariates
beta <- c(0.5, 1)
p.occ <- length(beta)
# Detection covariates
alpha <- list()
for (i in 1:n.data) {
  alpha[[i]] <- runif(2, -1, 1)
}
p.det.long <- sapply(alpha, length)
p.det <- sum(p.det.long)

# Simulate occupancy data.
dat <- simIntOcc(n.data = n.data, J.x = J.x, J.y = J.y, J.obs = J.obs,
 n.rep = n.rep, beta = beta, alpha = alpha, sp = FALSE)

y <- dat$y
X <- dat$X.obs
X.p <- dat$X.p
sites <- dat$sites

# Package all data into a list
```

```
occ.covs <- X[, 2, drop = FALSE]
colnames(occ.covs) <- c('occ.cov')
det.covs <- list()
# Add covariates one by one
det.covs[[1]] <- list(det.cov.1.1 = X.p[[1]][, , 2])
det.covs[[2]] <- list(det.cov.2.1 = X.p[[2]][, , 2])
det.covs[[3]] <- list(det.cov.3.1 = X.p[[3]][, , 2])
det.covs[[4]] <- list(det.cov.4.1 = X.p[[4]][, , 2])
data.list <- list(y = y,
  occ.covs = occ.covs,
  det.covs = det.covs,
  sites = sites)

J <- length(dat$z.obs)
# Initial values
inits.list <- list(alpha = list(0, 0, 0, 0),
      beta = 0,
      z = rep(1, J))
# Priors
prior.list <- list(beta.normal = list(mean = 0, var = 2.72),
   alpha.normal = list(mean = list(0, 0, 0, 0),
               var = list(2.72, 2.72, 2.72, 2.72)))
n.samples <- 5000
out <- intPGOcc(occ.formula = ~ occ.cov,
                det.formula = list(f.1 = ~ det.cov.1.1,
                   f.2 = ~ det.cov.2.1,
                   f.3 = ~ det.cov.3.1,
                   f.4 = ~ det.cov.4.1),
data = data.list,
inits = inits.list,
n.samples = n.samples,
priors = prior.list,
n.omp.threads = 1,
verbose = TRUE,
n.report = 1000,
n.burn = 1000,
n.thin = 1)

summary(out)
```

`msPGOcc` *Function for Fitting Multispecies Occupancy Models Using Polya-Gamma Latent Variables*

Description

Function for fitting multispecies occupancy models using Polya-Gamma latent variables.

Usage

```
msPGOcc(occ.formula, det.formula, data, inits, priors, n.samples,
        n.omp.threads = 1, verbose = TRUE, n.report = 100,
        n.burn = round(.10 * n.samples), n.thin = 1,
        k.fold, k.fold.threads = 1, k.fold.seed, ...)
```

Arguments

occ.formula	a symbolic description of the model to be fit for the occurrence portion of the model using R's model syntax. Only right-hand side of formula is specified. See example below. Random intercepts are allowed using **lme4** syntax (Bates et al. 2015).
det.formula	a symbolic description of the model to be fit for the detection portion of the model using R's model syntax. Only right-hand side of formula is specified. See example below. Random intercepts are allowed using **lme4** syntax (Bates et al. 2015).
data	a list containing data necessary for model fitting. Valid tags are `y`, `occ.covs`, and `det.covs`. `y` is a three-dimensional array with first dimension equal to the number of species, second dimension equal to the number of sites, and third dimension equal to the maximum number of replicates at a given site. `occ.covs` is a matrix or data frame containing the variables used in the occurrence portion of the model, with J rows for each column (variable). `det.covs` is a list of variables included in the detection portion of the model. Each list element is a different detection covariate, which can be site-level or observational-level. Site-level covariates are specified as a vector of length J while observation-level covariates are specified as a matrix or data frame with the number of rows equal to J and number of columns equal to the maximum number of replicates at a given site.
inits	a list with each tag corresponding to a parameter name. Valid tags are `alpha.comm`, `beta.comm`, `beta`, `alpha`, `tau.sq.beta`, `tau.sq.alpha`, `z`. The value portion of each tag is the parameter's initial value. See `priors` description for definition of each parameter name.
priors	a list with each tag corresponding to a parameter name. Valid tags are `beta.comm.normal`, `alpha.comm.normal`, `tau.sq.beta.ig`, and `tau.sq.alpha.ig`. Community-level occurrence (`beta.comm`) and detection (`alpha.comm`) regression coefficients are assumed to follow a normal distribution. The hyperparameters of the normal distribution are passed as a list of length two with the first and second elements corresponding to the mean and variance of the normal distribution, which are each specified as vectors of length equal to the number of coefficients to be estimated or of length one if priors are the same for all coefficients. If not specified, prior means are set to 0 and prior variances set to 2.72. Community-level variance parameters for occurrence (`tau.sq.beta`) and detection (`tau.sq.alpha`) are assumed to follow an inverse Gamma distribution. The hyperparameters of the inverse gamma distribution are passed as a list of length two with the first and second elements corresponding to the shape and scale parameters, which are each specified as vectors of length equal to the number of

coefficients to be estimated or a single value if all parameters are assigned the same prior. If not specified, prior shape and scale parameters are set to 0.1.

`n.samples` the number of posterior samples to collect.

`n.omp.threads` a positive integer indicating the number of threads to use for SMP parallel processing. The package must be compiled for OpenMP support. For most Intel-based machines, we recommend setting `n.omp.threads` up to the number of hypterthreaded cores. Note, `n.omp.threads` > 1 might not work on some systems.

`verbose` if `TRUE`, messages about data preparation, model specification, and progress of the sampler are printed to the screen. Otherwise, no messages are printed.

`n.report` the interval to report MCMC progress.

`n.burn` the number of samples out of the total `n.samples` to discard as burn-in. By default, the first 10% of samples is discarded.

`n.thin` the thinning interval for collection of MCMC samples. The thinning occurs after the `n.burn` samples are discarded. Default value is set to 1.

`k.fold` specifies the number of k folds for cross-validation. If not specified as an argument, then cross-validation is not performed and `k.fold.threads` and `k.fold.seed` are ignored. In k-fold cross-validation, the data specified in `data` is randomly partitioned into k equal sized subsamples. Of the k subsamples, k - 1 subsamples are used to fit the model and the remaining k samples are used for prediction. The cross-validation process is repeated k times (the folds). As a scoring rule, we use the model deviance as described in Hooten and Hobbs (2015). Cross-validation is performed after the full model is fit using all the data. Cross-validation results are reported in the `k.fold.deviance` object in the return list.

`k.fold.threads` number of threads to use for cross-validation. If `k.fold.threads` > 1 parallel processing is accomplished using the **foreach** and **doParallel** packages. Ignored if `k.fold` is not specified.

`k.fold.seed` seed used to split data set into `k.fold` parts for k-fold cross-validation. Ignored if `k.fold` is not specified.

`...` currently no additional arguments

Value

An object of class `msPGOcc` that is a list comprised of:

`beta.comm.samples`
: a `coda` object of posterior samples for the community level occurrence regression coefficients.

`alpha.comm.samples`
: a `coda` object of posterior samples for the community level detection regression coefficients.

`tau.sq.beta.samples`
: a `coda` object of posterior samples for the occurrence community variance parameters.

`tau.sq.alpha.samples`
: a `coda` object of posterior samples for the detection community variance parameters.

`beta.samples` a coda object of posterior samples for the species level occurrence regression coefficients.

`alpha.samples` a coda object of posterior samples for the species level detection regression coefficients.

`z.samples` a three-dimensional array of posterior samples for the latent occurrence values for each species.

`psi.samples` a three-dimensional array of posterior samples for the latent occurrence probability values for each species.

`sigma.sq.psi.samples`
a coda object of posterior samples for variances of random intercepts included in the occurrence portion of the model. Only included if random intercepts are specified in `occ.formula`.

`sigma.sq.p.samples`
a coda object of posterior samples for variances of random intercepts included in the detection portion of the model. Only included if random intercepts are specified in `det.formula`.

`beta.star.samples`
a coda object of posterior samples for the occurrence random effects. Only included if random intercepts are specified in `occ.formula`.

`alpha.star.samples`
a coda object of posterior samples for the detection random effects. Only included if random intercepts are specified in `det.formula`.

`run.time` MCMC sampler execution time reported using `proc.time()`.

`k.fold.deviance`
vector of scoring rules (deviance) from k-fold cross-validation. A separate value is reported for each species. Only included if `k.fold` is specified in function call.

The return object will include additional objects used for subsequent prediction and/or model fit evaluation.

Note

Some of the underlying code used for generating random numbers from the Polya-Gamma distribution is taken from the **pgdraw** package written by Daniel F. Schmidt and Enes Makalic. Their code implements Algorithm 6 in PhD thesis of Jesse Bennett Windle (2013) https://repositories.lib.utexas.edu/handle/2152/21842.

Author(s)

Jeffrey W. Doser <doserjef@msu.edu>,
Andrew O. Finley <finleya@msu.edu>

References

Polson, N.G., J.G. Scott, and J. Windle. (2013) Bayesian Inference for Logistic Models Using Polya-Gamma Latent Variables. *Journal of the American Statistical Association*, 108:1339-1349.

Bates, Douglas, Martin Maechler, Ben Bolker, Steve Walker (2015). Fitting Linear Mixed-Effects Models Using lme4. Journal of Statistical Software, 67(1), 1-48. doi: 10.18637/jss.v067.i01.

Hooten, M. B., and Hobbs, N. T. (2015). A guide to Bayesian model selection for ecologists. Ecological monographs, 85(1), 3-28.

Dorazio, R. M., and Royle, J. A. (2005). Estimating size and composition of biological communities by modeling the occurrence of species. Journal of the American Statistical Association, 100(470), 389-398.

Examples

```
set.seed(400)
J.x <- 8
J.y <- 8
J <- J.x * J.y
n.rep<- sample(2:4, size = J, replace = TRUE)
N <- 6
# Community-level covariate effects
# Occurrence
beta.mean <- c(0.2, 0.5)
p.occ <- length(beta.mean)
tau.sq.beta <- c(0.6, 0.3)
# Detection
alpha.mean <- c(0.5, 0.2, -0.1)
tau.sq.alpha <- c(0.2, 0.3, 1)
p.det <- length(alpha.mean)
# Draw species-level effects from community means.
beta <- matrix(NA, nrow = N, ncol = p.occ)
alpha <- matrix(NA, nrow = N, ncol = p.det)
for (i in 1:p.occ) {
  beta[, i] <- rnorm(N, beta.mean[i], sqrt(tau.sq.beta[i]))
}
for (i in 1:p.det) {
  alpha[, i] <- rnorm(N, alpha.mean[i], sqrt(tau.sq.alpha[i]))
}

dat <- simMsOcc(J.x = J.x, J.y = J.y, n.rep = n.rep, N = N, beta = beta, alpha = alpha,
sp = FALSE)
y <- dat$y
X <- dat$X
X.p <- dat$X.p
# Package all data into a list
occ.covs <- X[, 2, drop = FALSE]
colnames(occ.covs) <- c('occ.cov')
det.covs <- list(det.cov.1 = X.p[, , 2],
 det.cov.2 = X.p[, , 3]
 )
data.list <- list(y = y,
  occ.covs = occ.covs,
  det.covs = det.covs)

# Occupancy initial values
prior.list <- list(beta.comm.normal = list(mean = 0, var = 2.72),
```

```
   alpha.comm.normal = list(mean = 0, var = 2.72),
   tau.sq.beta.ig = list(a = 0.1, b = 0.1),
   tau.sq.alpha.ig = list(a = 0.1, b = 0.1))
# Initial values
inits.list <- list(alpha.comm = 0,
   beta.comm = 0,
   beta = 0,
   alpha = 0,
   tau.sq.beta = 1,
   tau.sq.alpha = 1,
   z = apply(y, c(1, 2), max, na.rm = TRUE))

n.samples <- 3000
n.burn <- 2000
n.thin <- 1

out <- msPGOcc(occ.formula = ~ occ.cov,
       det.formula = ~ det.cov.1 + det.cov.2,
       data = data.list,
       inits = inits.list,
       n.samples = n.samples,
       priors = prior.list,
              n.omp.threads = 1,
       verbose = TRUE,
       n.report = 1000,
       n.burn = n.burn,
       n.thin = n.thin)

summary(out, level = 'community')
```

---

neon2015 *Detection-nondetection data of 12 foliage gleaning bird species in 2015 in Bartlett Experimental Forest in New Hampshire, USA*

Description

Detection-nondetection data of 12 foliage gleaning bird species in 2015 in the Bartlett Experimental Forest in New Hampshire, USA. These data were collected as part of the National Ecological Observatory Network (NEON). Data were collected at 80 sites where observers recorded the number of all bird species observed during a six minute, 125m radius point count survey once during the breeding season. The six minute survey was split into three two-minute intervals following a removal design where the observer recorded the interval during which a species was first observed (if any) with a 1, intervals prior to observation with a 0, and then mentally removed the species from subsequent intervals (marked with NA), which enables modeling of data in an occupancy modeling framework. The 12 species included in the data set are as follows: (1) AMRE: American Redstart; (2) BAWW: Black-and-white Warbler; (3) BHVI: Blue-headed Vireo; (4) BLBW: Blackburnian Warbler; (5) BLPW: Blackpoll Warbler; (6) BTBW: Black-throated Blue Warbler; (7) BTNW: BLack-throated Green Warbler; (8) CAWA: Canada Warbler; (9) MAWA: Magnolia Warbler; (10) NAWA: Nashville Warbler; (11) OVEN: Ovenbird; (12) REVI: Red-eyed Vireo.

Usage

```
data(neon2015)
```

Format

neon2015 is a list with four elements:

y: a three-dimensional array of detection-nondetection data with dimensions of species (12), sites (80) and replicates (3).

occ.covs: a numeric matrix with 80 rows and one column consisting of the elevation at each site.

det.covs: a list of two numeric vectors with 80 elements. The first element is the day of year when the survey was conducted for a given site. The second element is the time of day when the survey began.

coords: a numeric matrix with 80 rows and two columns containing the site coordinates (Easting and Northing) in UTM Zone 19. The proj4string is "+proj=utm +zone=19 +units=m +datum=NAD83".

Source

NEON (National Ecological Observatory Network). Breeding landbird point counts, RELEASE-2021 (DP1.10003.001). https://doi.org/10.48443/s730-dy13. Dataset accessed from https://data.neonscience.org on October 10, 2021

References

Doser, J.W., Leuenberger, W., Sillett, T.S., Hallworth, M.T., Zipkin, E.F. Integrated community occupancy models: A framework to assess occurrence and biodiversity dynamics usingmultiple data sources. https://arxiv.org/abs/2109.01894

Barnett, D. T., Duffy, P. A., Schimel, D. S., Krauss, R. E., Irvine, K. M., Davis, F. W.,Gross, J. E., Azuaje, E. I., Thorpe, A. S., Gudex-Cross, D., et al. (2019). The terrestrial organism and biogeochemistry spatial sampling design for the national ecological observatory network. Ecosphere, 10(2):e02540.

PGOcc *Function for Fitting Single Species Occupancy Models Using Polya-Gamma Latent Variables*

Description

Function for fitting single species occupancy models using Polya-Gamma latent variables.

Usage

```
PGOcc(occ.formula, det.formula, data, inits, priors, n.samples,
      n.omp.threads = 1, verbose = TRUE, n.report = 100,
      n.burn = round(.10 * n.samples), n.thin = 1,
      k.fold, k.fold.threads = 1, k.fold.seed, ...)
```

Arguments

occ.formula	a symbolic description of the model to be fit for the occurrence portion of the model using R's model syntax. Only right-hand side of formula is specified. See example below. Random intercepts are allowed using lme4 syntax (Bates et al. 2015).
det.formula	a symbolic description of the model to be fit for the detection portion of the model using R's model syntax. Only right-hand side of formula is specified. See example below. Random intercepts are allowed using lme4 syntax (Bates et al. 2015).
data	a list containing data necessary for model fitting. Valid tags are y, occ.covs, and det.covs. y is a matrix or data frame with first dimension equal to the number of sites (J) and second dimension equal to the maximum number of replicates at a given site. occ.covs is a matrix or data frame containing the variables used in the occurrence portion of the model, with J rows for each column (variable). det.covs is a list of variables included in the detection portion of the model. Each list element is a different detection covariate, which can be site-level or observational-level. Site-level covariates are specified as a vector of length J while observation-level covariates are specified as a matrix or data frame with the number of rows equal to J and number of columns equal to the maximum number of replicates at a given site.
inits	a list with each tag corresponding to a parameter name. Valid tags are z, beta, alpha, sigma.sq.psi, and sigma.sq.p. The value portion of each tag is the parameter's initial value. sigma.sq.psi and sigma.sq.p are only relevant when including random effects in the occurrence and detection portion of the occupancy model, respectively. See priors description for definition of each parameter name.
priors	a list with each tag corresponding to a parameter name. Valid tags are beta.normal, alpha.normal, sigma.sq.psi.ig, and sigma.sq.p.ig. Occupancy (beta) and detection (alpha) regression coefficients are assumed to follow a normal distribution. The hyperparameters of the normal distribution are passed as a list of length two with the first and second elements corresponding to the mean and variance of the normal distribution, which are each specified as vectors of length equal to the number of coefficients to be estimated or of length one if priors are the same for all coefficients. If not specified, prior means are set to 0 and prior variances set to 2.72. sigma.sq.psi and sigma.sq.p are the random effect variances for any occurrence or detection random effects, respectively, and are assumed to follow an inverse Gamma distribution. The hyperparameters of the inverse-Gamma distribution are passed as a list of length two with first and second elements corresponding to the shape and scale parameters, respectively, which are each specified as vectors of length equal to the number of random intercepts or of length one if priors are the same for all random effect variances.
n.samples	the number of posterior samples to collect.
n.omp.threads	a positive integer indicating the number of threads to use for SMP parallel processing. The package must be compiled for OpenMP support. For most Intel-based machines, we recommend setting n.omp.threads up to the number of hypterthreaded cores. Note, n.omp.threads > 1 might not work on some systems.

verbose	if TRUE, messages about data preparation, model specification, and progress of the sampler are printed to the screen. Otherwise, no messages are printed.
n.report	the interval to report MCMC progress.
n.burn	the number of samples out of the total n.samples to discard as burn-in. By default, the first 10% of samples is discarded.
n.thin	the thinning interval for collection of MCMC samples. The thinning occurs after the n.burn samples are discarded. Default value is set to 1.
k.fold	specifies the number of k folds for cross-validation. If not specified as an argument, then cross-validation is not performed and k.fold.threads and k.fold.seed are ignored. In k-fold cross-validation, the data specified in data is randomly partitioned into k equal sized subsamples. Of the k subsamples, k - 1 subsamples are used to fit the model and the remaining k samples are used for prediction. The cross-validation process is repeated k times (the folds). As a scoring rule, we use the model deviance as described in Hooten and Hobbs (2015). Cross-validation is performed after the full model is fit using all the data. Cross-validation results are reported in the k.fold.deviance object in the return list.
k.fold.threads	number of threads to use for cross-validation. If k.fold.threads > 1 parallel processing is accomplished using the **foreach** and **doParallel** packages. Ignored if k.fold is not specified.
k.fold.seed	seed used to split data set into k.fold parts for k-fold cross-validation. Ignored if k.fold is not specified.
...	currently no additional arguments

Value

An object of class PGOcc that is a list comprised of:

beta.samples	a coda object of posterior samples for the occupancy regression coefficients.
alpha.samples	a coda object of posterior samples for the detection regression coefficients.
z.samples	a coda object of posterior samples for the latent occupancy values
psi.samples	a coda object of posterior samples for the latent occupancy probability values
sigma.sq.psi.samples	a coda object of posterior samples for variances of random intercepts included in the occupancy portion of the model. Only included if random intercepts are specified in occ.formula.
sigma.sq.p.samples	a coda object of posterior samples for variances of random intercpets included in the detection portion of the model. Only included if random intercepts are specified in det.formula.
beta.star.samples	a coda object of posterior samples for the occurrence random effects. Only included if random intercepts are specified in occ.formula.
alpha.star.samples	a coda object of posterior samples for the detection random effects. Only included if random intercepts are specified in det.formula.

`run.time` execution time reported using `proc.time()`.

`k.fold.deviance` scoring rule (deviance) from k-fold cross-validation. Only included if `k.fold` is specified in function call.

The return object will include additional objects used for subsequent prediction and/or model fit evaluation.

Note

Some of the underlying code used for generating random numbers from the Polya-Gamma distribution is taken from the **pgdraw** package written by Daniel F. Schmidt and Enes Makalic. Their code implements Algorithm 6 in PhD thesis of Jesse Bennett Windle (2013) https://repositories.lib.utexas.edu/handle/2152/21842.

Author(s)

Jeffrey W. Doser <doserjef@msu.edu>,
Andrew O. Finley <finleya@msu.edu>

References

Polson, N.G., J.G. Scott, and J. Windle. (2013) Bayesian Inference for Logistic Models Using Polya-Gamma Latent Variables. *Journal of the American Statistical Association*, 108:1339-1349.

Bates, Douglas, Martin Maechler, Ben Bolker, Steve Walker (2015). Fitting Linear Mixed-Effects Models Using lme4. Journal of Statistical Software, 67(1), 1-48. doi: 10.18637/jss.v067.i01.

Hooten, M. B., and Hobbs, N. T. (2015). A guide to Bayesian model selection for ecologists. Ecological monographs, 85(1), 3-28.

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.

Examples

```
set.seed(400)
J.x <- 10
J.y <- 10
J <- J.x * J.y
n.rep <- sample(2:4, J, replace = TRUE)
beta <- c(0.5, -0.15)
p.occ <- length(beta)
alpha <- c(0.7, 0.4)
p.det <- length(alpha)
dat <- simOcc(J.x = J.x, J.y = J.y, n.rep = n.rep, beta = beta, alpha = alpha,
      sp = FALSE)
occ.covs <- dat$X[, 2, drop = FALSE]
colnames(occ.covs) <- c('occ.cov')
det.covs <- list(det.cov = dat$X.p[, , 2])
# Data bundle
data.list <- list(y = dat$y,
```

```
  occ.covs = occ.covs,
  det.covs = det.covs)

# Priors
prior.list <- list(beta.normal = list(mean = 0, var = 2.72),
   alpha.normal = list(mean = 0, var = 2.72))
# Initial values
inits.list <- list(alpha = 0, beta = 0,
      z = apply(data.list$y, 1, max, na.rm = TRUE))

n.samples <- 5000
n.report <- 1000

out <- PGOcc(occ.formula = ~ occ.cov,
     det.formula = ~ det.cov,
     data = data.list,
     inits = inits.list,
     n.samples = n.samples,
     priors = prior.list,
     n.omp.threads = 1,
     verbose = TRUE,
     n.report = n.report,
     n.burn = 1000,
     n.thin = 1)
summary(out)
```

---

ppcOcc *Function for performing posterior predictive checks*

Description

Function for performing posterior predictive checks on spOccupancy model objects.

Usage

```
ppcOcc(object, fit.stat, group, ...)
```

Arguments

object	an object of class PGOcc, spPGOcc, msPGOcc, spMsPGOcc, intPGOcc, or spIntPGOcc.
fit.stat	a quoted keyword that specifies the fit statistic to use in the posterior predictive check. Supported fit statistics are "freeman-tukey" and "chi-square".
group	a positive integer indicating the way to group the detection-nondetection data for the posterior predictive check. Value 1 will group values by row (site) and value 2 will group values by column (replicate).
...	currently no additional arguments

Details

Standard GoF assessments are not valid for binary data, and posterior predictive checks must be performed on some sort of binned data.

Value

An object of class ppcOcc that is a list comprised of:

`fit.y` a numeric vector of posterior samples for the fit statistic calculated on the observed data.

`fit.y.rep` a numeric vector of posterior samples for the fit statistic calculated on a replicate data set generated from the model.

`fit.y.group.quants` a matrix consisting of posterior quantiles for the fit statistic using the observed data for each unique element the fit statistic is calculated for (i.e., sites when group = 1, replicates when group = 2).

`fit.y.rep.group.quants` a matrix consisting of posterior quantiles for the fit statistic using the model replicated data for each unique element the fit statistic is calculated for (i.e., sites when group = 1, replicates when group = 2).

The return object will include additional objects used for standard extractor functions.

Author(s)

Jeffrey W. Doser <doserjef@msu.edu>,
Andrew O. Finley <finleya@msu.edu>

Examples

```
set.seed(400)
# Simulate Data ------------------------------------------------------------
J.x <- 8
J.y <- 8
J <- J.x * J.y
n.rep <- sample(2:4, J, replace = TRUE)
beta <- c(0.5, -0.15)
p.occ <- length(beta)
alpha <- c(0.7, 0.4)
p.det <- length(alpha)
dat <- simOcc(J.x = J.x, J.y = J.y, n.rep = n.rep, beta = beta, alpha = alpha,
      sp = FALSE)
occ.covs <- dat$X[, 2, drop = FALSE]
colnames(occ.covs) <- c('occ.cov')
det.covs <- list(det.cov = dat$X.p[, , 2])
# Data bundle
data.list <- list(y = dat$y,
  occ.covs = occ.covs,
  det.covs = det.covs)
```

```
# Priors
prior.list <- list(beta.normal = list(mean = 0, var = 2.72),
   alpha.normal = list(mean = 0, var = 2.72))
# Initial values
inits.list <- list(alpha = 0, beta = 0,
      z = apply(data.list$y, 1, max, na.rm = TRUE))

n.samples <- 5000
n.report <- 1000

out <- PGOcc(occ.formula = ~ occ.cov,
     det.formula = ~ det.cov,
     data = data.list,
     inits = inits.list,
     n.samples = n.samples,
     priors = prior.list,
     n.omp.threads = 1,
     verbose = TRUE,
     n.report = n.report,
     n.burn = 4000,
     n.thin = 1)

# Posterior predictive check
ppc.out <- ppcOcc(out, fit.stat = 'chi-square', group = 1)
summary(ppc.out)
```

`predict.intPGOcc` *Function for prediction at new locations for single species integrated occupancy models*

Description

The function `predict` collects posterior predictive samples for a set of new locations given an object of class 'intPGOcc'.

Usage

```
## S3 method for class 'intPGOcc'
predict(object, X.0, ...)
```

Arguments

`object`	an object of class intPGOcc
`X.0`	the design matrix for prediction locations. This should include a column of 1s for the intercept. Covariates should have the same column names as those used when fitting the model with `intPGOcc`.
`...`	currently no additional arguments

Value

An object of class `predict.intPGOcc` that is a list comprised of:

`psi.0.samples`	a coda object of posterior predictive samples for the latent occurrence probability values.
`z.0.samples`	a coda object of posterior predictive samples for the latent occurrence values.

The return object will include additional objects used for standard extractor functions.

Author(s)

Jeffrey W. Doser <doserjef@msu.edu>,
Andrew O. Finley <finleya@msu.edu>

Examples

```
set.seed(1008)

# Simulate Data ------------------------------------------------------------
J.x <- 10
J.y <- 10
J.all <- J.x * J.y
# Number of data sources.
n.data <- 4
# Sites for each data source.
J.obs <- sample(ceiling(0.2 * J.all):ceiling(0.5 * J.all), n.data, replace = TRUE)
# Replicates for each data source.
n.rep <- list()
for (i in 1:n.data) {
  n.rep[[i]] <- sample(1:4, size = J.obs[i], replace = TRUE)
}
# Occupancy covariates
beta <- c(0.5, 1)
p.occ <- length(beta)
# Detection covariates
alpha <- list()
for (i in 1:n.data) {
  alpha[[i]] <- runif(2, -1, 1)
}
p.det.long <- sapply(alpha, length)
p.det <- sum(p.det.long)

# Simulate occupancy data.
dat <- simIntOcc(n.data = n.data, J.x = J.x, J.y = J.y, J.obs = J.obs,
 n.rep = n.rep, beta = beta, alpha = alpha, sp = FALSE)

y <- dat$y
X <- dat$X.obs
X.p <- dat$X.p
sites <- dat$sites
```

```
# Package all data into a list
occ.covs <- X[, 2, drop = FALSE]
colnames(occ.covs) <- c('occ.cov')
det.covs <- list()
# Add covariates one by one
det.covs[[1]] <- list(det.cov.1.1 = X.p[[1]][, , 2])
det.covs[[2]] <- list(det.cov.2.1 = X.p[[2]][, , 2])
det.covs[[3]] <- list(det.cov.3.1 = X.p[[3]][, , 2])
det.covs[[4]] <- list(det.cov.4.1 = X.p[[4]][, , 2])
data.list <- list(y = y,
  occ.covs = occ.covs,
  det.covs = det.covs,
  sites = sites)

J <- length(dat$z.obs)
# Initial values
inits.list <- list(alpha = list(0, 0, 0, 0),
      beta = 0,
      z = rep(1, J))
# Priors
prior.list <- list(beta.normal = list(mean = 0, var = 2.72),
   alpha.normal = list(mean = list(0, 0, 0, 0),
               var = list(2.72, 2.72, 2.72, 2.72)))
n.samples <- 5000
out <- intPGOcc(occ.formula = ~ occ.cov,
                det.formula = list(f.1 = ~ det.cov.1.1,
                   f.2 = ~ det.cov.2.1,
                   f.3 = ~ det.cov.3.1,
                   f.4 = ~ det.cov.4.1),
data = data.list,
inits = inits.list,
n.samples = n.samples,
priors = prior.list,
n.omp.threads = 1,
verbose = TRUE,
n.report = 1000,
n.burn = 4000,
n.thin = 1)

summary(out)

# Prediction
X.0 <- dat$X.pred
psi.0 <- dat$psi.pred

out.pred <- predict(out, X.0)
psi.hat.quants <- apply(out.pred$psi.0.samples, 2, quantile, c(0.025, 0.5, 0.975))
plot(psi.0, psi.hat.quants[2, ], pch = 19, xlab = 'True',
     ylab = 'Fitted', ylim = c(min(psi.hat.quants), max(psi.hat.quants)))
segments(psi.0, psi.hat.quants[1, ], psi.0, psi.hat.quants[3, ])
lines(psi.0, psi.0)
```

`predict.msPGOcc` *Function for prediction at new locations for multispecies occupancy models*

Description

The function `predict` collects posterior predictive samples for a set of new locations given an object of class 'msPGOcc'. When non-spatial random effects are included in the occurrence portion of the model, `predict` only allows prediction at random effect levels that are included in the fitted data.

Usage

```
## S3 method for class 'msPGOcc'
predict(object, X.0, ...)
```

Arguments

`object`	an object of class msPGOcc
`X.0`	the design matrix for prediction locations. This should include a column of 1s for the intercept. If random effects are included in the occurrence portion of the model, the levels of the random effects at the new locations should be included as a column in the design matrix. Covariates should have the same column names as those used when fitting the model with `msPGOcc`.
`...`	currently no additional arguments

Value

An object of class `predict.msPGOcc` that is a list comprised of:

`psi.0.samples`	a three-dimensional array of posterior predictive samples for the latent occurrence probability values.
`z.0.samples`	a three-dimensional array of posterior predictive samples for the latent occurrence values.

The return object will include additional objects used for standard extractor functions.

Author(s)

Jeffrey W. Doser <doserjef@msu.edu>,
Andrew O. Finley <finleya@msu.edu>

Examples

```
set.seed(400)
J.x <- 8
J.y <- 8
J <- J.x * J.y
n.rep<- sample(2:4, size = J, replace = TRUE)
N <- 6
# Community-level covariate effects
# Occurrence
beta.mean <- c(0.2, 0.5)
p.occ <- length(beta.mean)
tau.sq.beta <- c(0.6, 0.3)
# Detection
alpha.mean <- c(0.5, 0.2, -0.1)
tau.sq.alpha <- c(0.2, 0.3, 1)
p.det <- length(alpha.mean)
# Draw species-level effects from community means.
beta <- matrix(NA, nrow = N, ncol = p.occ)
alpha <- matrix(NA, nrow = N, ncol = p.det)
for (i in 1:p.occ) {
  beta[, i] <- rnorm(N, beta.mean[i], sqrt(tau.sq.beta[i]))
}
for (i in 1:p.det) {
  alpha[, i] <- rnorm(N, alpha.mean[i], sqrt(tau.sq.alpha[i]))
}

dat <- simMsOcc(J.x = J.x, J.y = J.y, n.rep = n.rep, N = N, beta = beta, alpha = alpha,
sp = FALSE)
n.samples <- 5000
# Split into fitting and prediction data set
pred.indx <- sample(1:J, round(J * .25), replace = FALSE)
y <- dat$y[, -pred.indx, ]
# Occupancy covariates
X <- dat$X[-pred.indx, ]
# Detection covariates
X.p <- dat$X.p[-pred.indx, , ]
# Prediction values
X.0 <- dat$X[pred.indx, ]
psi.0 <- dat$psi[, pred.indx]
# Package all data into a list
occ.covs <- X[, 2, drop = FALSE]
colnames(occ.covs) <- c('occ.cov')
det.covs <- list(det.cov.1 = X.p[, , 2],
 det.cov.2 = X.p[, , 3]
 )
data.list <- list(y = y,
  occ.covs = occ.covs,
  det.covs = det.covs)

# Occupancy initial values
prior.list <- list(beta.comm.normal = list(mean = 0, var = 2.72),
   alpha.comm.normal = list(mean = 0, var = 2.72),
```

```
   tau.sq.beta.ig = list(a = 0.1, b = 0.1),
   tau.sq.alpha.ig = list(a = 0.1, b = 0.1))
# Initial values
inits.list <- list(alpha.comm = 0,
      beta.comm = 0,
      beta = 0,
      alpha = 0,
      tau.sq.beta = 1,
      tau.sq.alpha = 1,
      z = apply(y, c(1, 2), max, na.rm = TRUE))

out <- msPGOcc(occ.formula = ~ occ.cov,
       det.formula = ~ det.cov.1 + det.cov.2,
       data = data.list,
       inits = inits.list,
       n.samples = n.samples,
       priors = prior.list,
              n.omp.threads = 1,
       verbose = TRUE,
       n.report = 1000,
       n.burn = 4000)

summary(out, level = 'community')

# Predict at new locations ---------------------------------------------
out.pred <- predict(out, X.0)
```

`predict.PGOcc` *Function for prediction at new locations for single species occupancy models*

Description

The function `predict` collects posterior predictive samples for a set of new locations given an object of class 'PGOcc'. When non-spatial random effects are included in the occurrence portion of the model, `predict` only allows prediction at random effect levels that are included in the fitted data.

Usage

```
## S3 method for class 'PGOcc'
predict(object, X.0, ...)
```

Arguments

`object`	an object of class PGOcc
`X.0`	the design matrix for prediction locations. This should include a column of 1s for the intercept. If random effects are included in the occurrence portion of the model, the levels of the random effects at the new locations should be included as a column in the design matrix. The ordering of the levels should match the

ordering used to fit the data in PGOcc. Covariates should have the same column names as those used when fitting the model with PGOcc.

... currently no additional arguments

Value

An object of class predict.PGOcc that is a list comprised of:

psi.0.samples a coda object of posterior predictive samples for the latent occupancy probability values.

z.0.samples a coda object of posterior predictive samples for the latent occupancy values.

The return object will include additional objects used for standard extractor functions.

Author(s)

Jeffrey W. Doser <doserjef@msu.edu>,
Andrew O. Finley <finleya@msu.edu>

Examples

```
set.seed(400)
# Simulate Data ------------------------------------------------------------
J.x <- 10
J.y <- 10
J <- J.x * J.y
n.rep <- sample(2:4, J, replace = TRUE)
beta <- c(0.5, 2)
p.occ <- length(beta)
alpha <- c(0, 1)
p.det <- length(alpha)
dat <- simOcc(J.x = J.x, J.y = J.y, n.rep = n.rep, beta = beta, alpha = alpha,
      sp = FALSE)
# Split into fitting and prediction data set
pred.indx <- sample(1:J, round(J * .25), replace = FALSE)
y <- dat$y[-pred.indx, ]
# Occupancy covariates
X <- dat$X[-pred.indx, ]
# Prediction covariates
X.0 <- dat$X[pred.indx, ]
# Detection covariates
X.p <- dat$X.p[-pred.indx, , ]

# Package all data into a list
occ.covs <- X[, 2, drop = FALSE]
colnames(occ.covs) <- c('occ.cov')
det.covs <- list(det.cov = X.p[, , 2])
data.list <- list(y = y,
  occ.covs = occ.covs,
  det.covs = det.covs)
# Priors
prior.list <- list(beta.normal = list(mean = rep(0, p.occ),
```

```
      var = rep(2.72, p.occ)),
   alpha.normal = list(mean = rep(0, p.det),
               var = rep(2.72, p.det)))
# Initial values
inits.list <- list(alpha = rep(0, p.det),
      beta = rep(0, p.occ),
      z = apply(y, 1, max, na.rm = TRUE))

n.samples <- 5000
n.report <- 1000

out <- PGOcc(occ.formula = ~ occ.cov,
     det.formula = ~ det.cov,
     data = data.list,
     inits = inits.list,
     n.samples = n.samples,
     priors = prior.list,
     n.omp.threads = 1,
     verbose = TRUE,
     n.report = n.report,
     n.burn = 4000,
     n.thin = 1)

summary(out)

# Predict at new locations ------------------------------------------------

out.pred <- predict(out, X.0)
psi.0.quants <- apply(out.pred$psi.0.samples, 2, quantile, c(0.025, 0.5, 0.975))
plot(dat$psi[pred.indx], psi.0.quants[2, ], pch = 19, xlab = 'True',
     ylab = 'Fitted', ylim = c(min(psi.0.quants), max(psi.0.quants)))
segments(dat$psi[pred.indx], psi.0.quants[1, ], dat$psi[pred.indx], psi.0.quants[3, ])
lines(dat$psi[pred.indx], dat$psi[pred.indx])
```

`predict.spIntPGOcc` *Function for prediction at new locations for single species integrated spatial occupancy models*

Description

The function `predict` collects posterior predictive samples for a set of new locations given an object of class 'spIntPGOcc'.

Usage

```
## S3 method for class 'spIntPGOcc'
predict(object, X.0, coords.0, n.omp.threads = 1, verbose = TRUE,
                              n.report = 100, ...)
```

Arguments

object	an object of class spIntPGOcc.
X.0	the design matrix for prediction locations. This should include a column of 1s for the intercept. Covariates should have the same column names as those used when fitting the model with spIntPGOcc.
coords.0	the spatial coordinates corresponding to X.0.
n.omp.threads	a positive integer indicating the number of threads to use for SMP parallel processing. The package must be compiled for OpenMP support. For most Intel-based machines, we recommend setting n.omp.threads up to the number of hyperthreaded cores. Note, n.omp.threads > 1 might not work on some systems.
verbose	if TRUE, model specification and progress of the sampler is printed to the screen. Otherwise, nothing is printed to the screen.
n.report	the interval to report sampling progress.
...	currently no additional arguments

Value

An object of class predict.spIntPGOcc that is a list comprised of:

psi.0.samples	a coda object of posterior predictive samples for the latent occurrence probability values.
z.0.samples	a coda object of posterior predictive samples for the latent occurrence values.

The return object will include additional objects used for standard extractor functions.

Author(s)

Jeffrey W. Doser <doserjef@msu.edu>,
Andrew O. Finley <finleya@msu.edu>

References

Hooten, M. B., and Hefley, T. J. (2019). Bringing Bayesian models to life. CRC Press.

Examples

```
set.seed(400)

# Simulate Data ----------------------------------------------------------
# Number of locations in each direction. This is the total region of interest
# where some sites may or may not have a data source.
J.x <- 8
J.y <- 8
J.all <- J.x * J.y
# Number of data sources.
n.data <- 4
# Sites for each data source.
```

```
J.obs <- sample(ceiling(0.2 * J.all):ceiling(0.5 * J.all), n.data, replace = TRUE)
# Replicates for each data source.
n.rep <- list()
for (i in 1:n.data) {
  n.rep[[i]] <- sample(1:4, size = J.obs[i], replace = TRUE)
}
# Occupancy covariates
beta <- c(0.5, 0.5)
p.occ <- length(beta)
# Detection covariates
alpha <- list()
alpha[[1]] <- runif(2, 0, 1)
alpha[[2]] <- runif(3, 0, 1)
alpha[[3]] <- runif(2, -1, 1)
alpha[[4]] <- runif(4, -1, 1)
p.det.long <- sapply(alpha, length)
p.det <- sum(p.det.long)
sigma.sq <- 2
phi <- 3 / .5
sp <- TRUE

# Simulate occupancy data.
dat <- simIntOcc(n.data = n.data, J.x = J.x, J.y = J.y, J.obs = J.obs,
 n.rep = n.rep, beta = beta, alpha = alpha, sp = sp,
 phi = phi, sigma.sq = sigma.sq, cov.model = 'spherical')

y <- dat$y
X <- dat$X.obs
X.p <- dat$X.p
sites <- dat$sites
X.0 <- dat$X.pred
psi.0 <- dat$psi.pred
coords <- as.matrix(dat$coords.obs)
coords.0 <- as.matrix(dat$coords.pred)

# Package all data into a list
occ.covs <- X[, 2, drop = FALSE]
colnames(occ.covs) <- c('occ.cov')
det.covs <- list()
# Add covariates one by one
det.covs[[1]] <- list(det.cov.1.1 = X.p[[1]][, , 2])
det.covs[[2]] <- list(det.cov.2.1 = X.p[[2]][, , 2],
      det.cov.2.2 = X.p[[2]][, , 3])
det.covs[[3]] <- list(det.cov.3.1 = X.p[[3]][, , 2])
det.covs[[4]] <- list(det.cov.4.1 = X.p[[4]][, , 2],
      det.cov.4.2 = X.p[[4]][, , 3],
      det.cov.4.3 = X.p[[4]][, , 4])
data.list <- list(y = y,
  occ.covs = occ.covs,
  det.covs = det.covs,
  sites = sites,
  coords = coords)
```

```
J <- length(dat$z.obs)

# Initial values
inits.list <- list(alpha = list(0, 0, 0, 0),
      beta = 0,
      phi = 3 / .5,
      sigma.sq = 2,
      w = rep(0, J),
      z = rep(1, J))
# Priors
prior.list <- list(beta.normal = list(mean = 0, var = 2.72),
   alpha.normal = list(mean = list(0, 0, 0, 0),
       var = list(2.72, 2.72, 2.72, 2.72)),
   phi.unif = c(3/1, 3/.1),
   sigma.sq.ig = c(2, 2))
# Tuning
tuning.list <- list(phi = 1)

# Number of batches
n.batch <- 40
# Batch length
batch.length <- 25

out <- spIntPGOcc(occ.formula = ~ occ.cov,
  det.formula = list(f.1 = ~ det.cov.1.1,
     f.2 = ~ det.cov.2.1 + det.cov.2.2,
     f.3 = ~ det.cov.3.1,
     f.4 = ~ det.cov.4.1 + det.cov.4.2 + det.cov.4.3),
  data = data.list,
  inits = inits.list,
  n.batch = n.batch,
  batch.length = batch.length,
  accept.rate = 0.43,
  priors = prior.list,
  cov.model = "spherical",
  tuning = tuning.list,
  n.omp.threads = 1,
  verbose = TRUE,
  NNGP = TRUE,
  n.neighbors = 5,
  search.type = 'cb',
  n.report = 10,
  n.burn = 500,
  n.thin = 1)
summary(out)

# Predict at new locations ---------------------------------------------
out.pred <- predict(out, X.0, coords.0, verbose = FALSE)
```

predict.spMsPGOcc *Function for prediction at new locations for multispecies spatial occupancy models*

Description

The function predict collects posterior predictive samples for a set of new locations given an object of class 'spMsPGOcc'.

Usage

```
## S3 method for class 'spMsPGOcc'
predict(object, X.0, coords.0, n.omp.threads = 1, verbose = TRUE,
                            n.report = 100, ...)
```

Arguments

object	an object of class spMsPGOcc
X.0	the design matrix for prediction locations. This should include a column of 1s for the intercept. Covariates should have the same column names as those used when fitting the model with intPGOcc.
coords.0	the spatial coordinates corresponding to X.0.
n.omp.threads	a positive integer indicating the number of threads to use for SMP parallel processing. The package must be compiled for OpenMP support. For most Intel-based machines, we recommend setting n.omp.threads up to the number of hyperthreaded cores. Note, n.omp.threads > 1 might not work on some systems.
verbose	if TRUE, model specification and progress of the sampler is printed to the screen. Otherwise, nothing is printed to the screen.
n.report	the interval to report sampling progress.
...	currently no additional arguments

Value

An object of class predict.spMsPGOcc that is a list comprised of:

psi.0.samples	a three-dimensional array of posterior predictive samples for the latent occurrence probability values.
z.0.samples	a three-dimensional array of posterior predictive samples for the latent occurrence values.
w.0.samples	a three-dimensional array of posterior predictive samples for the latent spatial random effects.
run.time	execution time reported using proc.time().

The return object will include additional objects used for standard extractor functions.

Author(s)

Jeffrey W. Doser <doserjef@msu.edu>,
Andrew O. Finley <finleya@msu.edu>

Examples

```
set.seed(400)

# Simulate Data ------------------------------------------------------------
J.x <- 7
J.y <- 7
J <- J.x * J.y
n.rep <- sample(2:4, size = J, replace = TRUE)
N <- 5
# Community-level covariate effects
# Occurrence
beta.mean <- c(0.2, -0.15)
p.occ <- length(beta.mean)
tau.sq.beta <- c(0.6, 0.3)
# Detection
alpha.mean <- c(0.5, 0.2, -.2)
tau.sq.alpha <- c(0.2, 0.3, 0.8)
p.det <- length(alpha.mean)
# Draw species-level effects from community means.
beta <- matrix(NA, nrow = N, ncol = p.occ)
alpha <- matrix(NA, nrow = N, ncol = p.det)
for (i in 1:p.occ) {
  beta[, i] <- rnorm(N, beta.mean[i], sqrt(tau.sq.beta[i]))
}
for (i in 1:p.det) {
  alpha[, i] <- rnorm(N, alpha.mean[i], sqrt(tau.sq.alpha[i]))
}
phi <- runif(N, 3/1, 3/.4)
sigma.sq <- runif(N, 0.3, 3)
sp <- TRUE

dat <- simMsOcc(J.x = J.x, J.y = J.y, n.rep = n.rep, N = N, beta = beta, alpha = alpha,
phi = phi, sigma.sq = sigma.sq, sp = TRUE, cov.model = 'exponential')

# Number of batches
n.batch <- 40
# Batch length
batch.length <- 25
n.samples <- n.batch * batch.length

# Split into fitting and prediction data set
pred.indx <- sample(1:J, round(J * .25), replace = FALSE)
y <- dat$y[, -pred.indx, ]
# Occupancy covariates
X <- dat$X[-pred.indx, ]
# Coordinates
coords <- as.matrix(dat$coords[-pred.indx, ])
```

```
# Detection covariates
X.p <- dat$X.p[-pred.indx, , ]
# Prediction values
X.0 <- dat$X[pred.indx, ]
coords.0 <- as.matrix(dat$coords[pred.indx, ])
psi.0 <- dat$psi[, pred.indx]

# Package all data into a list
occ.covs <- X[, 2, drop = FALSE]
colnames(occ.covs) <- c('occ.cov')
det.covs <- list(det.cov.1 = X.p[, , 2],
 det.cov.2 = X.p[, , 3]
 )
data.list <- list(y = y,
  occ.covs = occ.covs,
  det.covs = det.covs,
  coords = coords)

# Priors
prior.list <- list(beta.comm.normal = list(mean = 0, var = 2.72),
   alpha.comm.normal = list(mean = 0, var = 2.72),
   tau.sq.beta.ig = list(a = 0.1, b = 0.1),
   tau.sq.alpha.ig = list(a = 0.1, b = 0.1),
   phi.unif = list(a = 3/1, b = 3/.1),
   sigma.sq.ig = list(a = 2, b = 2))
# Starting values
inits.list <- list(alpha.comm = 0,
      beta.comm = 0,
      beta = 0,
      alpha = 0,
      tau.sq.beta = 1,
      tau.sq.alpha = 1,
      phi = 3 / .5,
      sigma.sq = 2,
      w = matrix(0, nrow = N, ncol = nrow(X)),
      z = apply(y, c(1, 2), max, na.rm = TRUE))
# Tuning
tuning.list <- list(phi = 1)

out <- spMsPGOcc(occ.formula = ~ occ.cov,
 det.formula = ~ det.cov.1 + det.cov.2,
 data = data.list,
 inits = inits.list,
 n.batch = n.batch,
 batch.length = batch.length,
 accept.rate = 0.43,
         priors = prior.list,
 cov.model = "exponential",
 tuning = tuning.list,
         n.omp.threads = 1,
         verbose = TRUE,
 NNGP = TRUE,
 n.neighbors = 5,
```

```
 search.type = 'cb',
        n.report = 10,
 n.burn = 500,
 n.thin = 1)

summary(out, level = 'both')

# Predict at new locations ------------------------------------------------
out.pred <- predict(out, X.0, coords.0, verbose = FALSE)
```

`predict.spPGOcc` *Function for prediction at new locations for single species spatial occupancy models*

Description

The function `predict` collects posterior predictive samples for a set of new locations given an object of class 'spPGOcc'.

Usage

```
## S3 method for class 'spPGOcc'
predict(object, X.0, coords.0, n.omp.threads = 1, verbose = TRUE,
                          n.report = 100, ...)
```

Arguments

`object`	an object of class `spPGOcc`
`X.0`	the design matrix for prediction locations. This should include a column of 1s for the intercept if an intercept is included in the original model fit. Covariates should have the same column names as those used when fitting the model with `spPGOcc`.
`coords.0`	the spatial coordinates corresponding to `X.0`.
`n.omp.threads`	a positive integer indicating the number of threads to use for SMP parallel processing. The package must be compiled for OpenMP support. For most Intel-based machines, we recommend setting `n.omp.threads` up to the number of hyperthreaded cores. Note, `n.omp.threads` > 1 might not work on some systems.
`verbose`	if `TRUE`, model specification and progress of the sampler is printed to the screen. Otherwise, nothing is printed to the screen.
`n.report`	the interval to report sampling progress.
`...`	currently no additional arguments

Value

An object of class `predict.spPGOcc` that is a list comprised of:

`psi.0.samples`	a coda object of posterior predictive samples for the latent occurrence probability values.
`z.0.samples`	a coda object of posterior predictive samples for the latent occurrence values.
`w.0.samples`	a coda object of posterior predictive samples for the latent spatial random effects.
`run.time`	execution time reported using `proc.time()`.

The return object will include additional objects used for standard extractor functions.

Author(s)

Jeffrey W. Doser <doserjef@msu.edu>,
Andrew O. Finley <finleya@msu.edu>

References

Hooten, M. B., and Hefley, T. J. (2019). Bringing Bayesian models to life. CRC Press.

Examples

```
set.seed(400)
# Simulate Data ------------------------------------------------------------
J.x <- 8
J.y <- 8
J <- J.x * J.y
n.rep <- sample(2:4, J, replace = TRUE)
beta <- c(0.5, 2)
p.occ <- length(beta)
alpha <- c(0, 1)
p.det <- length(alpha)
phi <- 3 / .6
sigma.sq <- 2
dat <- simOcc(J.x = J.x, J.y = J.y, n.rep = n.rep, beta = beta, alpha = alpha,
              sigma.sq = sigma.sq, phi = phi, sp = TRUE, cov.model = 'exponential')
# Split into fitting and prediction data set
pred.indx <- sample(1:J, round(J * .5), replace = FALSE)
y <- dat$y[-pred.indx, ]
# Occupancy covariates
X <- dat$X[-pred.indx, ]
# Prediction covariates
X.0 <- dat$X[pred.indx, ]
# Detection covariates
X.p <- dat$X.p[-pred.indx, , ]
coords <- as.matrix(dat$coords[-pred.indx, ])
coords.0 <- as.matrix(dat$coords[pred.indx, ])
psi.0 <- dat$psi[pred.indx]
w.0 <- dat$w[pred.indx]
```

```
# Package all data into a list
occ.covs <- X[, -1, drop = FALSE]
colnames(occ.covs) <- c('occ.cov')
det.covs <- list(det.cov.1 = X.p[, , 2])
data.list <- list(y = y,
  occ.covs = occ.covs,
  det.covs = det.covs,
  coords = coords)

# Number of batches
n.batch <- 50
# Batch length
batch.length <- 25
n.iter <- n.batch * batch.length
# Priors
prior.list <- list(beta.normal = list(mean = 0, var = 2.72),
   alpha.normal = list(mean = 0, var = 2.72),
   sigma.sq.ig = c(2, 2),
   phi.unif = c(3/1, 3/.1))
# Initial values
inits.list <- list(alpha = 0, beta = 0,
      phi = 3 / .5,
      sigma.sq = 2,
      w = rep(0, nrow(X)),
      z = apply(y, 1, max, na.rm = TRUE))
# Tuning
tuning.list <- list(phi = 1)

out <- spPGOcc(occ.formula = ~ occ.cov,
       det.formula = ~ det.cov.1,
       data = data.list,
       inits = inits.list,
       n.batch = n.batch,
       batch.length = batch.length,
       accept.rate = 0.43,
       priors = prior.list,
       cov.model = 'exponential',
       tuning = tuning.list,
       n.omp.threads = 1,
       verbose = TRUE,
       NNGP = TRUE,
       n.neighbors = 15,
       search.type = 'cb',
       n.report = 10,
       n.burn = 500,
       n.thin = 1)

summary(out)

# Predict at new locations ---------------------------------------------------
out.pred <- predict(out, X.0, coords.0, verbose = FALSE)
```

simIntOcc *Simulate Single-Species Detection-Nondetection Data from Multiple Data Sources*

Description

The function simIntOcc simulates single-species detection-nondetection data from multiple data sources for simulation studies, power assessments, or function testing of integrated occupancy models. Data can optionally be simulated with a spatial Gaussian Process on the occurrence process.

Usage

```
simIntOcc(n.data, J.x, J.y, J.obs, n.rep, beta, alpha,
          sp = FALSE, cov.model, sigma.sq, phi, nu, ...)
```

Arguments

n.data	an integer indicating the number of detection-nondetection data sources to simulate.
J.x	a single numeric value indicating the number of sites across the region of interest along the horizontal axis. Total number of sites across the simulated region of interest is $J.x \times J.y$.
J.y	a single numeric value indicating the number of sites across the region of interest along the vertical axis. Total number of sites across the simulated region of interest is $J.x \times J.y$.
J.obs	a numeric vector of length n.data containing the number of sites to simulate each data source at. Data sources can be obtained at completely different sites, the same sites, or anywhere inbetween. Maximum number of sites a given data source is available at is equal to $J = J.x \times J.y$.
n.rep	a list of length n.data. Each element is a numeric vector with length corresponding to the number of sites that given data source is observed at (in J.obs). Each vector indicates the number of repeat visits at each of the sites for a given data source.
beta	a numeric vector containing the intercept and regression coefficient parameters for the occurrence portion of the single-species occupancy model.
alpha	a list of length n.data. Each element is a numeric vector containing the intercept and regression coefficient parameters for the detection portion of the single-species occupancy model for each data source.
sp	a logical value indicating whether to simulate a spatially-explicit occupancy model with a Gaussian process and exponential correlation function. By default set to FALSE.
cov.model	a quoted keyword that specifies the covariance function used to model the spatial dependence structure among the latent occurrence values. Supported covariance model key words are: "exponential", "matern", "spherical", and "gaussian".

sigma.sq	a numeric value indicating the spatial variance parameter. Ignored when sp = FALSE.
phi	a numeric value indicating the spatial range parameter. Ignored when sp = FALSE.
nu	a numeric value indicating the spatial smoothness parameter. Only used when sp = TRUE and cov.model = "matern".
...	currently no additional arguments

Value

A list comprised of:

X.obs	a numeric design matrix for the occurrence portion of the model. This matrix contains the intercept and regression coefficients for only the observed sites.
X.pred	a numeric design matrix for the occurrence portion of the model at sites where there are no observed data sources.
X.p	a list of design matrices for the detection portions of the integrated occupancy model. Each element in the list is a design matrix of detection covariates for each data source.
coords.obs	a numeric matrix of coordinates of each observed site. Required for spatial models.
coords.pred	a numeric matrix of coordinates of each site in the study region without any data sources. Only used for spatial models.
D.obs	a distance matrix of observed sites. Only used for spatial models.
D.pred	a distance matrix of sites in the study region without any observed data. Only used for spatial models.
w.obs	a matrix of the spatial random effects at observed locations. Only used to simulate data when sp = TRUE
.	
w.pred	a matrix of the spatial random random effects at locations without any observation.
psi.obs	a matrix of the occurrence probabilities for each observed site.
psi.pred	a matrix of the occurrence probabilities for sites without any observations.
z.obs	a vector of the latent occurrence states at each observed site.
z.pred	a vector of the latent occurrence states at each site without any observations.
p	a list of detection probability matrices for each of the n.data data sources.
y	a list of matrices of the raw detection-nondetection data for each site and replicate combination.

Author(s)

Jeffrey W. Doser <doserjef@msu.edu>,
Andrew O. Finley <finleya@msu.edu>

Examples

```
set.seed(400)

# Simulate Data ------------------------------------------------------------
J.x <- 15
J.y <- 15
J.all <- J.x * J.y
# Number of data sources.
n.data <- 4
# Sites for each data source.
J.obs <- sample(ceiling(0.2 * J.all):ceiling(0.5 * J.all), n.data, replace = TRUE)
# Replicates for each data source.
n.rep <- list()
for (i in 1:n.data) {
  n.rep[[i]] <- sample(1:4, size = J.obs[i], replace = TRUE)
}
# Occupancy covariates
beta <- c(0.5, 1, -3)
p.occ <- length(beta)
# Detection covariates
alpha <- list()
for (i in 1:n.data) {
  alpha[[i]] <- runif(sample(1:4, 1), -1, 1)
}
p.det.long <- sapply(alpha, length)
p.det <- sum(p.det.long)
sigma.sq <- 2
phi <- 3 / .5
sp <- TRUE

# Simulate occupancy data.
dat <- simIntOcc(n.data = n.data, J.x = J.x, J.y = J.y, J.obs = J.obs,
 n.rep = n.rep, beta = beta, alpha = alpha, sp = TRUE,
 cov.model = 'gaussian', sigma.sq = sigma.sq, phi = phi)
```

---

simMsOcc *Simulate Multi-Species Detection-Nondetection Data*

Description

The function simMsOcc simulates multi-species detection-nondetection data for simulation studies, power assessments, or function testing. Data can be optionally simulated with a spatial Gaussian Process in the occurrence portion of the model. Non-spatial random intercepts can also be included in the detection or occurrence portions of the occupancy model.

Usage

```
simMsOcc(J.x, J.y, n.rep, N, beta, alpha, psi.RE = list(),
         p.RE = list(), sp = FALSE, cov.model, sigma.sq, phi, nu, ...)
```

Arguments

J.x	a single numeric value indicating the number of sites to simulate detection-nondetection data along the horizontal axis. Total number of sites with simulated data is $J.x \times J.y$.
J.y	a single numeric value indicating the number of sites to simulate detection-nondetection data along the vertical axis. Total number of sites with simulated data is $J.x \times J.y$.
n.rep	a numeric vector of length $J = J.x \times J.y$ indicating the number of repeat visits at each of the J sites.
N	a single numeric value indicating the number of species to simulate detection-nondetection data.
beta	a numeric matrix with N rows containing the intercept and regression coefficient parameters for the occurrence portion of the multispecies occupancy model. Each row corresponds to the regression coefficients for a given species.
alpha	a numeric matrix with N rows containing the intercept and regression coefficient parameters for the detection portion of the multispecies occupancy model. Each row corresponds to the regression coefficients for a given species.
psi.RE	a list used to specify the non-spatial random intercepts included in the occurrence portion of the model. The list must have two tags: `levels` and `sigma.sq.psi`. `levels` is a vector of length equal to the number of distinct random intercepts to include in the model and contains the number of levels there are in each intercept. `sigma.sq.psi` is a vector of length equal to the number of distinct random intercepts to include in the model and contains the variances for each random effect. If not specified, no random effects are included in the occurrence portion of the model.
p.RE	a list used to specify the non-spatial random intercepts included in the detection portion of the model. The list must have two tags: `levels` and `sigma.sq.p`. `levels` is a vector of length equal to the number of distinct random intercepts to include in the model and contains the number of levels there are in each intercept. `sigma.sq.p` is a vector of length equal to the number of distinct random intercepts to include in the model and contains the variances for each random effect. If not specified, no random effects are included in the detection portion of the model.
sp	a logical value indicating whether to simulate a spatially-explicit occupancy model with a Gaussian process and exponential correlation function. By default set to `FALSE`.
cov.model	a quoted keyword that specifies the covariance function used to model the spatial dependence structure among the latent occurrence values. Supported covariance model key words are: `"exponential"`, `"matern"`, `"spherical"`, and `"gaussian"`.
sigma.sq	a numeric vector of length N containing the spatial variance parameter for each species. Ignored when `sp = FALSE`.
phi	a numeric vector of length N containing the spatial range parameter for each species. Ignored when `sp = FALSE`.

nu	a numeric vector of length N containing the spatial smoothness parameter for each species. Only used when sp = TRUE and cov.model = 'matern'.
...	currently no additional arguments

Value

A list comprised of:

X	a $J \times p.occ$ numeric design matrix for the occurrence portion of the model.
X.p	a three-dimensional numeric array with dimensions corresponding to sites, repeat visits, and number of detection regression coefficients. This is the design matrix used for the detection portion of the occupancy model.
coords	a $J \times J$ numeric matrix of coordinates of each occupancy site. Required for spatial models.
w	a $N \times J$ matrix of the spatial random effects for each species. Only used to simulate data when sp = TRUE.
psi	a $N \times J$ matrix of the occurrence probabilities for each species at each site.
z	a $N \times J$ matrix of the latent occurrence states for each species at each site.
p	a N x J x max(n.rep) array of the detection probabilities for each species at each site and replicate combination. Sites with fewer than max(n.rep) replicates will contain NA values.
y	a N x J x max(n.rep) array of the raw detection-nondetection data for each species at each site and replicate combination. Sites with fewer than max(n.rep) replicates will contain NA values.
X.p.re	a three-dimensional numeric array containing the levels of any detection random effect included in the model. Only relevant when detection random effects are specified in p.RE.
X.lambda.re	a numeric matrix containing the levels of any occurrence random effect included in the model. Only relevant when occurrence random effects are specified in psi.RE.
alpha.star	a numeric matrix where each row contains the simulated detection random effects for each given level of the random effects included in the detection model. Only relevant when detection random effects are included in the model.
beta.star	a numeric matrix where each row contains the simulated occurrence random effects for each given level of the random effects included in the occurrence model. Only relevant when occurrence random effects are included in the model.

Author(s)

Jeffrey W. Doser <doserjef@msu.edu>,
Andrew O. Finley <finleya@msu.edu>

Examples

```
J.x <- 8
J.y <- 8
J <- J.x * J.y
n.rep <- sample(2:4, size = J, replace = TRUE)
N <- 10
# Community-level covariate effects
# Occurrence
beta.mean <- c(0.2, -0.15)
p.occ <- length(beta.mean)
tau.sq.beta <- c(0.6, 0.3)
# Detection
alpha.mean <- c(0.5, 0.2)
tau.sq.alpha <- c(0.2, 0.3)
p.det <- length(alpha.mean)
psi.RE <- list(levels = c(10),
       sigma.sq.psi = c(1.5))
p.RE <- list(levels = c(15),
     sigma.sq.p = 0.8)
# Draw species-level effects from community means.
beta <- matrix(NA, nrow = N, ncol = p.occ)
alpha <- matrix(NA, nrow = N, ncol = p.det)
for (i in 1:p.occ) {
  beta[, i] <- rnorm(N, beta.mean[i], sqrt(tau.sq.beta[i]))
}
for (i in 1:p.det) {
  alpha[, i] <- rnorm(N, alpha.mean[i], sqrt(tau.sq.alpha[i]))
}
# Spatial parameters if desired
phi <- runif(N, 3/1, 3/.1)
sigma.sq <- runif(N, 0.3, 3)
sp <- TRUE

dat <- simMsOcc(J.x = J.x, J.y = J.y, n.rep = n.rep, N = N, beta = beta,
alpha = alpha, psi.RE = psi.RE, p.RE = p.RE, sp = TRUE,
cov.model = 'exponential', phi = phi, sigma.sq = sigma.sq)
```

`simOcc` *Simulate Single-Species Detection-Nondetection Data*

Description

The function `simOcc` simulates single-species occurrence data for simulation studies, power assessments, or function testing. Data can be optionally simulated with a spatial Gaussian Process in the occurrence portion of the model. Non-spatial random intercepts can also be included in the detection or occurrence portions of the occupancy model.

Usage

```
simOcc(J.x, J.y, n.rep, beta, alpha, psi.RE = list(),
       p.RE = list(), sp = FALSE, cov.model, sigma.sq, phi, nu, ...)
```

Arguments

J.x	a single numeric value indicating the number of sites to simulate detection-nondetection data along the horizontal axis. Total number of sites with simulated data is $J.x \times J.y$.
J.y	a single numeric value indicating the number of sites to simulate detection-nondetection data along the vertical axis. Total number of sites with simulated data is $J.x \times J.y$.
n.rep	a numeric vector of length $J = J.x \times J.y$ indicating the number of repeat visits at each of the J sites.
beta	a numeric vector containing the intercept and regression coefficient parameters for the occupancy portion of the single-species occupancy model.
alpha	a numeric vector containing the intercept and regression coefficient parameters for the detection portion of the single-species occupancy model.
psi.RE	a list used to specify the non-spatial random intercepts included in the occupancy portion of the model. The list must have two tags: `levels` and `sigma.sq.psi`. `levels` is a vector of length equal to the number of distinct random intercepts to include in the model and contains the number of levels there are in each intercept. `sigma.sq.psi` is a vector of length equal to the number of distinct random intercepts to include in the model and contains the variances for each random effect. If not specified, no random effects are included in the occupancy portion of the model.
p.RE	a list used to specify the non-spatial random intercepts included in the detection portion of the model. The list must have two tags: `levels` and `sigma.sq.p`. `levels` is a vector of length equal to the number of distinct random intercepts to include in the model and contains the number of levels there are in each intercept. `sigma.sq.p` is a vector of length equal to the number of distinct random intercepts to include in the model and contains the variances for each random effect. If not specified, no random effects are included in the detection portion of the model.
sp	a logical value indicating whether to simulate a spatially-explicit occupancy model with a Gaussian process and exponential correlation function. By default set to `FALSE`.
cov.model	a quoted keyword that specifies the covariance function used to model the spatial dependence structure among the latent occurrence values. Supported covariance model key words are: `"exponential"`, `"matern"`, `"spherical"`, and `"gaussian"`.
sigma.sq	a numeric value indicating the spatial variance parameter. Ignored when `sp = FALSE`.
phi	a numeric value indicating the spatial range parameter. Ignored when `sp = FALSE`.
nu	a numeric value indicating the spatial smoothness parameter. Only used when `sp = TRUE` and `cov.model = "matern"`.
...	currently no additional arguments

Value

A list comprised of:

X	a $J \times p.occ$ numeric design matrix for the occupancy portion of the model.
X.p	a three-dimensional numeric array with dimensions corresponding to sites, repeat visits, and number of detection regression coefficients. This is the design matrix used for the detection portion of the occupancy model.
coords	a $J \times J$ numeric matrix of coordinates of each occupancy site. Required for spatial models.
w	a $J \times 1$ matrix of the spatial random effects. Only used to simulate data when sp = TRUE.
psi	a $J \times 1$ matrix of the occupancy probabilities for each site.
z	a length J vector of the latent occupancy states at each site.
p	a J x max(n.rep) matrix of the detection probabilities for each site and replicate combination. Sites with fewer than max(n.rep) replicates will contain NA values.
y	a J x max(n.rep) matrix of the raw detection-nondetection data for each site and replicate combination.
X.p.re	a three-dimensional numeric array containing the levels of any detection random effect included in the model. Only relevant when detection random effects are specified in p.RE.
X.lambda.re	a numeric matrix containing the levels of any occurrence random effect included in the model. Only relevant when occurrence random effects are specified in psi.RE.
alpha.star	a numeric vector that contains the simulated detection random effects for each given level of the random effects included in the detection model. Only relevant when detection random effects are included in the model.
beta.star	a numeric vector that contains the simulated occurrence random effects for each given level of the random effects included in the occurrence model. Only relevant when occurrence random effects are included in the model.

Author(s)

Jeffrey W. Doser <doserjef@msu.edu>,
Andrew O. Finley <finleya@msu.edu>

Examples

```
set.seed(400)
J.x <- 10
J.y <- 10
n.rep <- rep(4, J.x * J.y)
beta <- c(0.5, -0.15)
alpha <- c(0.7, 0.4)
phi <- 3 / .6
sigma.sq <- 2
```

```
psi.RE <- list(levels = 10,
       sigma.sq.psi = 1.2)
p.RE <- list(levels = 15,
     sigma.sq.p = 0.8)
dat <- simOcc(J.x = J.x, J.y = J.y, n.rep = n.rep, beta = beta, alpha = alpha,
      psi.RE = psi.RE, p.RE = p.RE, sp = TRUE, cov.model = 'spherical',
      sigma.sq = sigma.sq, phi = phi)
```

`spIntPGOcc` *Function for Fitting Single Species Integrated Spatial Occupancy Models Using Polya-Gamma Latent Variables*

Description

The function `spIntPGOcc` fits single species integrated spatial occupancy models using Polya-Gamma latent variables. Models can be fit using either a full Gaussian process or a Nearest Neighbor Gaussian Process for large data sets. Data integration is done using a joint likelihood framework, assuming distinct detection models for each data source that are each conditional on a single latent occupancy process.

Usage

```
spIntPGOcc(occ.formula, det.formula, data, inits, priors,
           tuning, cov.model = "exponential", NNGP = TRUE,
           n.neighbors = 15, search.type = 'cb', n.batch,
           batch.length, accept.rate = 0.43, n.omp.threads = 1,
           verbose = TRUE, n.report = 100,
           n.burn = round(.10 * n.batch * batch.length),
           n.thin = 1, k.fold, k.fold.threads = 1,
           k.fold.seed, k.fold.data, ...)
```

Arguments

`occ.formula`	a symbolic description of the model to be fit for the occurrence portion of the model using R's model syntax. Only right-hand side of formula is specified. See example below.
`det.formula`	a list of symbolic descriptions of the models to be fit for the detection portion of the model using R's model syntax for each data set. Each element in the list is a formula for the detection model of a given data set. Only right-hand side of formula is specified. See example below.
`data`	a list containing data necessary for model fitting. Valid tags are `y`, `occ.covs`, `det.covs`, `sites` and `coords`. `y` is a list of matrices or data frames for each data set used in the integrated model. Each element of the list has first dimension equal to the number of sites with that data source and second dimension equal to the maximum number of replicates at a given site. `occ.covs` is a matrix or data frame containing the variables used in the occurrence portion of the model, with the number of rows being the number of sites with at least one

data source for each column (variable). `det.covs` is a list of variables included in the detection portion of the model for each data source. `det.covs` should have the same number of elements as `y`, where each element is itself a list. Each element of the list for a given data source is a different detection covariate, which can be site-level or observational-level. Site-level covariates are specified as a vector with length equal to the number of observed sites of that data source, while observation-level covariates are specified as a matrix or data frame with the number of rows equal to the number of observed sites of that data source and number of columns equal to the maximum number of replicates at a given site. `coords` is a matrix of the observation site coordinates.

`inits` a list with each tag corresponding to a parameter name. Valid tags are `z`, `beta`, `alpha`, `sigma.sq`, `phi`, `w`, and `nu`. The value portion of all tags except `alpha` is the parameter's initial value. The tag `alpha` is a list comprised of the initial values for the detection parameters for each data source. Each element of the list should be a vector of initial values for all detection parameters in the given data source or a single value for each data source to assign all parameters for a given data source the same initial value. See `priors` description for definition of each parameter name.

`priors` a list with each tag corresponding to a parameter name. Valid tags are `beta.normal`, `alpha.normal`, `phi.unif`, `sigma.sq.ig`, and `nu.unif`. Occurrence (`beta`) and detection (`alpha`) regression coefficients are assumed to follow a normal distribution. For `beta` hyperparameters of the normal distribution are passed as a list of length two with the first and second elements corresponding to the mean and variance of the normal distribution, which are each specified as vectors of length equal to the number of coefficients to be estimated or of length one if priors are the same for all coefficients. For the detection coefficients `alpha`, the mean and variance hyperparameters are themselves passed in as lists, with each element of the list corresponding to the specific hyperparameters for the detection parameters in a given data source. If not specified, prior means are set to 0 and prior variances set to 2.73 for normal priors. The spatial variance parameter, `sigma.sq`, is assumed to follow an inverse-Gamma distribution, whereas the spatial decay `phi` and smoothness `nu` parameters are assumed to follow Uniform distributions. The hyperparameters of the inverse-Gamma are passed as a vector of length two, with the first and second elements corresponding to the *shape* and *scale*, respectively. The hyperparameters of the Uniform are also passed as a vector of length two with the first and second elements corresponding to the lower and upper support, respectively.

`tuning` a list with each tag corresponding to a parameter name. Valid tags are `phi` and `nu`. The value portion of each tag defines the initial variance of the Adaptive sampler. See Roberts and Rosenthal (2009) for details.

`cov.model` a quoted keyword that specifies the covariance function used to model the spatial dependence structure among the observations. Supported covariance model key words are: `"exponential"`, `"matern"`, `"spherical"`, and `"gaussian"`.

`NNGP` if `TRUE`, model is fit with an NNGP. If `FALSE`, a full Gaussian process is used. See Datta et al. (2016) and Finley et al. (2019) for more information.

`n.neighbors` number of neighbors used in the NNGP. Only used if `NNGP = TRUE`. Datta et al. (2016) showed that 15 neighbors is usually sufficient, but that as few as 5

neighbors can be adequate for certain data sets, which can lead to even greater decreases in run time. We recommend starting with 15 neighbors (the default) and if additional gains in computation time are desired, subsequently compare the results with a smaller number of neighbors using WAIC or k-fold cross-validation.

`search.type` a quoted keyword that specifies the type of nearest neighbor search algorithm. Supported method key words are: `"cb"` and `"brute"`. The `"cb"` should generally be much faster. If locations do not have identical coordinate values on the axis used for the nearest neighbor ordering then `"cb"` and `"brute"` should produce identical neighbor sets. However, if there are identical coordinate values on the axis used for nearest neighbor ordering, then `"cb"` and `"brute"` might produce different, but equally valid, neighbor sets, e.g., if data are on a grid.

`n.batch` the number of MCMC batches to run for the Adaptive MCMC sampler. See Roberts and Rosenthal (2009) for details.

`batch.length` the length of each MCMC batch to run for the Adaptive MCMC sampler. See Roberts and Rosenthal (2009) for details.

`accept.rate` target acceptance rate for Adaptive MCMC. Default is 0.43. See Roberts and Rosenthal (2009) for details.

`n.omp.threads` a positive integer indicating the number of threads to use for SMP parallel processing. The package must be compiled for OpenMP support. For most Intel-based machines, we recommend setting `n.omp.threads` up to the number of hyperthreaded cores. Note, `n.omp.threads` > 1 might not work on some systems.

`verbose` if `TRUE`, messages about data preparation, model specification, and progress of the sampler are printed to the screen. Otherwise, no messages are printed.

`n.report` the interval to report Metropolis sampler acceptance and MCMC progress. Note this is specified in terms of batches and not overall samples for spatial models.

`n.burn` the number of samples out of the total `n.batch` * `batch.length` samples to discard as burn-in. By default, the first 10% of samples is discarded.

`n.thin` the thinning interval for collection of MCMC samples. The thinning occurs after the `n.burn` samples are discarded. Default value is set to 1.

`k.fold` specifies the number of k folds for cross-validation. If not specified as an argument, then cross-validation is not performed and `k.fold.threads` and `k.fold.seed` are ignored. In k-fold cross-validation, the data specified in `data` is randomly partitioned into k equal sized subsamples. Of the k subsamples, k - 1 subsamples are used to fit the model and the remaining k samples are used for prediction. The cross-validation process is repeated k times (the folds). As a scoring rule, we use the model deviance as described in Hooten and Hobbs (2015). Cross-validation is performed after the full model is fit using all the data. Cross-validation results are reported in the `k.fold.deviance` object in the return list.

`k.fold.threads` number of threads to use for cross-validation. If `k.fold.threads` > 1 parallel processing is accomplished using the **foreach** and **doParallel** packages. Ignored if `k.fold` is not specified.

`k.fold.seed` seed used to split data set into `k.fold` parts for k-fold cross-validation. Ignored if `k.fold` is not specified.

k.fold.data	an integer specifying the specific data set to hold out values from. If not specified, data from all data set locations will be incorporated into the k-fold cross-validation.
...	currently no additional arguments

Value

An object of class spIntPGOcc that is a list comprised of:

beta.samples	a coda object of posterior samples for the occurrence regression coefficients.
alpha.samples	a coda object of posterior samples for the detection regression coefficients for all data sources.
z.samples	a coda object of posterior samples for the latent occurrence values
psi.samples	a coda object of posterior samples for the latent occurrence probability values
y.rep.samples	a list of three-dimensional arrays of fitted values for each data source for use in Goodness of Fit assessments.
theta.samples	a coda object of posterior samples for covariance parameters.
w.samples	a coda object of posterior samples for latent spatial random effects.
run.time	execution time reported using proc.time().
k.fold.deviance	scoring rule (deviance) from k-fold cross-validation. A separate deviance value is returned for each data source. Only included if k.fold is specified in function call. Only a single value is returned if k.fold.data is specified.

The return object will include additional objects used for subsequent prediction and/or model fit evaluation.

Note

Some of the underlying code used for generating random numbers from the Polya-Gamma distribution is taken from the **pgdraw** package written by Daniel F. Schmidt and Enes Makalic. Their code implements Algorithm 6 in PhD thesis of Jesse Bennett Windle (2013) https://repositories.lib.utexas.edu/handle/2152/21842.

Author(s)

Jeffrey W. Doser <doserjef@msu.edu>,
Andrew O. Finley <finleya@msu.edu>

References

Datta, A., S. Banerjee, A.O. Finley, and A.E. Gelfand. (2016) Hierarchical Nearest-Neighbor Gaussian process models for large geostatistical datasets. *Journal of the American Statistical Association*, doi: 10.1080/01621459.2015.1044091.

Finley, A.O., A. Datta, B.D. Cook, D.C. Morton, H.E. Andersen, and S. Banerjee. (2019) Efficient algorithms for Bayesian Nearest Neighbor Gaussian Processes. *Journal of Computational and Graphical Statistics*, doi: 10.1080/10618600.2018.1537924.

Finley, A. O., Datta, A., and Banerjee, S. (2020). spNNGP R package for nearest neighbor Gaussian process models. *arXiv* preprint arXiv:2001.09111.

Hooten, M. B., and Hobbs, N. T. (2015). A guide to Bayesian model selection for ecologists. *Ecological Monographs*, 85(1), 3-28.

Hooten, M. B., and Hefley, T. J. (2019). Bringing Bayesian models to life. *CRC Press*.

Polson, N.G., J.G. Scott, and J. Windle. (2013) Bayesian Inference for Logistic Models Using Polya-Gamma Latent Variables. *Journal of the American Statistical Association*, 108:1339-1349.

Roberts, G.O. and Rosenthal J.S. (2009) Examples of adaptive MCMC. *Journal of Computational and Graphical Statistics*, 18(2):349-367.

Examples

```
set.seed(400)

# Simulate Data ------------------------------------------------------------
# Number of locations in each direction. This is the total region of interest
# where some sites may or may not have a data source.
J.x <- 8
J.y <- 8
J.all <- J.x * J.y
# Number of data sources.
n.data <- 4
# Sites for each data source.
J.obs <- sample(ceiling(0.2 * J.all):ceiling(0.5 * J.all), n.data, replace = TRUE)
# Replicates for each data source.
n.rep <- list()
for (i in 1:n.data) {
  n.rep[[i]] <- sample(1:4, size = J.obs[i], replace = TRUE)
}
# Occupancy covariates
beta <- c(0.5, 0.5)
p.occ <- length(beta)
# Detection covariates
alpha <- list()
alpha[[1]] <- runif(2, 0, 1)
alpha[[2]] <- runif(3, 0, 1)
alpha[[3]] <- runif(2, -1, 1)
alpha[[4]] <- runif(4, -1, 1)
p.det.long <- sapply(alpha, length)
p.det <- sum(p.det.long)
sigma.sq <- 2
phi <- 3 / .5
sp <- TRUE

# Simulate occupancy data from multiple data sources.
dat <- simIntOcc(n.data = n.data, J.x = J.x, J.y = J.y, J.obs = J.obs,
 n.rep = n.rep, beta = beta, alpha = alpha, sp = sp,
 sigma.sq = sigma.sq, phi = phi, cov.model = 'exponential')

y <- dat$y
X <- dat$X.obs
```

```
X.p <- dat$X.p
sites <- dat$sites
X.0 <- dat$X.pred
psi.0 <- dat$psi.pred
coords <- as.matrix(dat$coords.obs)
coords.0 <- as.matrix(dat$coords.pred)

# Package all data into a list
occ.covs <- X[, 2, drop = FALSE]
colnames(occ.covs) <- c('occ.cov')
det.covs <- list()
# Add covariates one by one
det.covs[[1]] <- list(det.cov.1.1 = X.p[[1]][, , 2])
det.covs[[2]] <- list(det.cov.2.1 = X.p[[2]][, , 2],
      det.cov.2.2 = X.p[[2]][, , 3])
det.covs[[3]] <- list(det.cov.3.1 = X.p[[3]][, , 2])
det.covs[[4]] <- list(det.cov.4.1 = X.p[[4]][, , 2],
      det.cov.4.2 = X.p[[4]][, , 3],
      det.cov.4.3 = X.p[[4]][, , 4])
data.list <- list(y = y,
  occ.covs = occ.covs,
  det.covs = det.covs,
  sites = sites,
  coords = coords)

J <- length(dat$z.obs)

# Initial values
inits.list <- list(alpha = list(0, 0, 0, 0),
      beta = 0,
      phi = 3 / .5,
      sigma.sq = 2,
      w = rep(0, J),
      z = rep(1, J))
# Priors
prior.list <- list(beta.normal = list(mean = 0, var = 2.72),
   alpha.normal = list(mean = list(0, 0, 0, 0),
       var = list(2.72, 2.72, 2.72, 2.72)),
   phi.unif = c(3/1, 3/.1),
   sigma.sq.ig = c(2, 2))
# Tuning
tuning.list <- list(phi = 0.3)

# Number of batches
n.batch <- 40
# Batch length
batch.length <- 25

out <- spIntPGOcc(occ.formula = ~ occ.cov,
  det.formula = list(f.1 = ~ det.cov.1.1,
     f.2 = ~ det.cov.2.1 + det.cov.2.2,
     f.3 = ~ det.cov.3.1,
     f.4 = ~ det.cov.4.1 + det.cov.4.2 + det.cov.4.3),
```

```
  data = data.list,
  inits = inits.list,
  n.batch = n.batch,
  batch.length = batch.length,
  accept.rate = 0.43,
  priors = prior.list,
  cov.model = "exponential",
  tuning = tuning.list,
  n.omp.threads = 1,
  verbose = TRUE,
  NNGP = TRUE,
  n.neighbors = 5,
  search.type = 'cb',
  n.report = 10,
  n.burn = 500,
  n.thin = 1)

summary(out)
```

---

spMsPGOcc *Function for Fitting Multispecies Spatial Occupancy Models Using Polya-Gamma Latent Variables*

Description

The function spMsPGOcc fits multispecies spatial occupancy models using Polya-Gamma latent variables. Models can be fit using either a full Gaussian process or a Nearest Neighbor Gaussian Process for large data sets.

Usage

```
spMsPGOcc(occ.formula, det.formula, data, inits, priors, tuning,
          cov.model = 'exponential', NNGP = TRUE,
          n.neighbors = 15, search.type = 'cb', n.batch,
          batch.length, accept.rate = 0.43, n.omp.threads = 1,
          verbose = TRUE, n.report = 100,
          n.burn = round(.10 * n.batch * batch.length), n.thin = 1,
          k.fold, k.fold.threads = 1, k.fold.seed, ...)
```

Arguments

occ.formula	a symbolic description of the model to be fit for the occurrence portion of the model using R's model syntax. Only right-hand side of formula is specified. See example below.
det.formula	a symbolic description of the model to be fit for the detection portion of the model using R's model syntax. Only right-hand side of formula is specified. See example below. Random intercepts are allowed using lme4 syntax (Bates et al. 2015).

`data` a list containing data necessary for model fitting. Valid tags are `y`, `occ.covs`, `det.covs`, `coords`. `y` is a three-dimensional array with first dimension equal to the number of species, second dimension equal to the number of sites, and third dimension equal to the maximum number of replicates at a given site. `occ.covs` is a matrix or data frame containing the variables used in the occurrence portion of the model, with J rows for each column (variable). `det.covs` is a list of variables included in the detection portion of the model. Each list element is a different detection covariate, which can be site-level or observational-level. Site-level covariates are specified as a vector of length J while observation-level covariates are specified as a matrix or data frame with the number of rows equal to J and number of columns equal to the maximum number of replicates at a given site. `coords` is a $J \times 2$ matrix of the observation coordinates.

`inits` a list with each tag corresponding to a parameter name. Valid tags are `alpha.comm`, `beta.comm`, `beta`, `alpha`, `tau.sq.beta`, `tau.sq.alpha`, `z`, `sigma.sq`, `phi`, `w`, and `nu`. `nu` is only specified if `cov.model = "matern"`. The value portion of each tag is the parameter's initial value. See `priors` description for definition of each parameter name.

`priors` a list with each tag corresponding to a parameter name. Valid tags are `beta.comm.normal`, `alpha.comm.normal`, `tau.sq.beta.ig`, `tau.sq.alpha.ig`, `phi.unif`, `sigma.sq.ig`, and `nu.unif`. Community-level occurrence (`beta.comm`) and detection (`alpha.comm`) regression coefficients are assumed to follow a normal distribution. The hyperparameters of the normal distribution are passed as a list of length two with the first and second elements corresponding to the mean and variance of the normal distribution, which are each specified as vectors of length equal to the number of coefficients to be estimated or of length one if priors are the same for all coefficients. If not specified, prior means are set to 0 and prior variances set to 2.73. Community-level variance parameters for occupancy (`tau.sq.beta`) and detection (`tau.sq.alpha`) are assumed to follow an inverse Gamma distribution. The hyperparameters of the inverse gamma distribution are passed as a list of length two with the first and second elements corresponding to the shape and scale parameters, which are each specified as vectors of length equal to the number of coefficients to be estimated or a single value if priors are the same for all parameters. If not specified, prior shape and scale parameters are set to 0.1. The species-specific spatial variance parameter, `sigma.sq`, is assumed to follow an inverse-Gamma distribution, whereas the spatial decay `phi` and smoothness `nu` parameters are assumed to follow Uniform distributions. The hyperparameters of the inverse-Gamma are passed as a list of length two, with the list elements being vectors of length N corresponding to the species-specific shape and scale parameters, respectively, or a single value if the same value is assigned for all species. The hyperparameters of the Uniform are also passed as a list with two elements, with both elements being vectors of length N corresponding to the lower and upper support, respectively, or as a single value if the same value is assigned for all species.

`tuning` a list with each tag corresponding to a parameter name. Valid tags are `phi` and `nu`. The value portion of each tag defines the initial variance of the adaptive sampler. We assume the initial variance of the adaptive sampler is the same for each species, although the adaptive sampler will adjust the tuning variances separately for each species. See Roberts and Rosenthal (2009) for details.

`cov.model` a quoted keyword that specifies the covariance function used to model the spatial dependence structure among the observations. Supported covariance model key words are: `"exponential"`, `"matern"`, `"spherical"`, and `"gaussian"`.

`NNGP` if `TRUE`, model is fit with an NNGP. If `FALSE`, a full Gaussian process is used. See Datta et al. (2016) and Finley et al. (2019) for more information.

`n.neighbors` number of neighbors used in the NNGP. Only used if `NNGP = TRUE`. Datta et al. (2016) showed that 15 neighbors is usually sufficient, but that as few as 5 neighbors can be adequate for certain data sets, which can lead to even greater decreases in run time. We recommend starting with 15 neighbors (the default) and if additional gains in computation time are desired, subsequently compare the results with a smaller number of neighbors using WAIC or k-fold cross-validation.

`search.type` a quoted keyword that specifies the type of nearest neighbor search algorithm. Supported method key words are: `"cb"` and `"brute"`. The `"cb"` should generally be much faster. If locations do not have identical coordinate values on the axis used for the nearest neighbor ordering then `"cb"` and `"brute"` should produce identical neighbor sets. However, if there are identical coordinate values on the axis used for nearest neighbor ordering, then `"cb"` and `"brute"` might produce different, but equally valid, neighbor sets, e.g., if data are on a grid.

`n.batch` the number of MCMC batches to run for the Adaptive MCMC sampler. See Roberts and Rosenthal (2009) for details.

`batch.length` the length of each MCMC batch to run for the Adaptive MCMC sampler. See Roberts and Rosenthal (2009) for details.

`accept.rate` target acceptance rate for Adaptive MCMC. Defaul is 0.43. See Roberts and Rosenthal (2009) for details.

`n.omp.threads` a positive integer indicating the number of threads to use for SMP parallel processing. The package must be compiled for OpenMP support. For most Intel-based machines, we recommend setting `n.omp.threads` up to the number of hyperthreaded cores. Note, `n.omp.threads` > 1 might not work on some systems.

`verbose` if `TRUE`, messages about data preparation, model specification, and progress of the sampler are printed to the screen. Otherwise, no messages are printed.

`n.report` the interval to report Metropolis sampler acceptance and MCMC progress. Note this is specified in terms of batches and not overall samples for spatial models.

`n.burn` the number of samples out of the total `n.samples` to discard as burn-in. By default, the first 10% of samples is discarded.

`n.thin` the thinning interval for collection of MCMC samples. The thinning occurs after the `n.burn` samples are discarded. Default value is set to 1.

`k.fold` specifies the number of k folds for cross-validation. If not specified as an argument, then cross-validation is not performed and `k.fold.threads` and `k.fold.seed` are ignored. In k-fold cross-validation, the data specified in `data` is randomly partitioned into k equal sized subsamples. Of the k subsamples, k - 1 subsamples are used to fit the model and the remaining k samples are used for prediction. The cross-validation process is repeated k times (the folds). As a scoring rule, we use the model deviance as described in Hooten and Hobbs (2015).

Cross-validation is performed after the full model is fit using all the data. Cross-validation results are reported in the `k.fold.deviance` object in the return list.

`k.fold.threads` number of threads to use for cross-validation. If `k.fold.threads > 1` parallel processing is accomplished using the **foreach** and **doParallel** packages. Ignored if `k.fold` is not specified.

`k.fold.seed` seed used to split data set into `k.fold` parts for k-fold cross-validation. Ignored if `k.fold` is not specified.

`...` currently no additional arguments

Value

An object of class `spMsPGOcc` that is a list comprised of:

`beta.comm.samples`
: a `coda` object of posterior samples for the community level occurrence regression coefficients.

`alpha.comm.samples`
: a `coda` object of posterior samples for the community level detection regression coefficients.

`tau.sq.beta.samples`
: a `coda` object of posterior samples for the occurrence community variance parameters.

`tau.sq.alpha.samples`
: a `coda` object of posterior samples for the detection community variance parameters.

`beta.samples` a `coda` object of posterior samples for the species level occurrence regression coefficients.

`alpha.samples` a `coda` object of posterior samples for the species level detection regression coefficients.

`theta.samples` a `coda` object of posterior samples for the species level covariance parameters.

`z.samples` a three-dimensional array of posterior samples for the latent occurrence values for each species.

`psi.samples` a three-dimensional array of posterior samples for the latent occupancy probability values for each species.

`w.samples` a three-dimensional array of posterior samples for the latent spatial random effects for each species.

`sigma.sq.p.samples`
: a `coda` object of posterior samples for variances of random intercpets included in the detection portion of the model. Only included if random intercepts are specified in `det.formula`.

`alpha.star.samples`
: a `coda` object of posterior samples for the detection random effects. Only included if random intercepts are specified in `det.formula`.

`run.time` MCMC sampler execution time reported using `proc.time()`.

`k.fold.deviance`
: vector of scoring rules (deviance) from k-fold cross-validation. A separate value is reported for each species. Only included if `k.fold` is specified in function call.

The return object will include additional objects used for subsequent prediction and/or model fit evaluation.

Note

Some of the underlying code used for generating random numbers from the Polya-Gamma distribution is taken from the **pgdraw** package written by Daniel F. Schmidt and Enes Makalic. Their code implements Algorithm 6 in PhD thesis of Jesse Bennett Windle (2013) https://repositories.lib.utexas.edu/handle/2152/21842.

Author(s)

Jeffrey W. Doser <doserjef@msu.edu>,
Andrew O. Finley <finleya@msu.edu>

References

Datta, A., S. Banerjee, A.O. Finley, and A.E. Gelfand. (2016) Hierarchical Nearest-Neighbor Gaussian process models for large geostatistical datasets. *Journal of the American Statistical Association*, doi: 10.1080/01621459.2015.1044091.

Finley, A.O., A. Datta, B.D. Cook, D.C. Morton, H.E. Andersen, and S. Banerjee. (2019) Efficient algorithms for Bayesian Nearest Neighbor Gaussian Processes. *Journal of Computational and Graphical Statistics*, doi: 10.1080/10618600.2018.1537924.

Finley, A. O., Datta, A., and Banerjee, S. (2020). spNNGP R package for nearest neighbor Gaussian process models. *arXiv* preprint arXiv:2001.09111.

Polson, N.G., J.G. Scott, and J. Windle. (2013) Bayesian Inference for Logistic Models Using Polya-Gamma Latent Variables. *Journal of the American Statistical Association*, 108:1339-1349.

Roberts, G.O. and Rosenthal J.S. (2009) Examples of adaptive MCMC. *Journal of Computational and Graphical Statistics*, 18(2):349-367.

Bates, Douglas, Martin Maechler, Ben Bolker, Steve Walker (2015). Fitting Linear Mixed-Effects Models Using lme4. Journal of Statistical Software, 67(1), 1-48. doi: 10.18637/jss.v067.i01.

Hooten, M. B., and Hobbs, N. T. (2015). A guide to Bayesian model selection for ecologists. *Ecological Monographs*, 85(1), 3-28.

Examples

```
set.seed(400)

# Simulate Data ------------------------------------------------------------
J.x <- 7
J.y <- 7
J <- J.x * J.y
n.rep <- sample(2:4, size = J, replace = TRUE)
```

```
N <- 5
# Community-level covariate effects
# Occurrence
beta.mean <- c(0.2, -0.15)
p.occ <- length(beta.mean)
tau.sq.beta <- c(0.6, 0.3)
# Detection
alpha.mean <- c(0.5, 0.2, -.2)
tau.sq.alpha <- c(0.2, 0.3, 0.8)
p.det <- length(alpha.mean)
# Draw species-level effects from community means.
beta <- matrix(NA, nrow = N, ncol = p.occ)
alpha <- matrix(NA, nrow = N, ncol = p.det)
for (i in 1:p.occ) {
  beta[, i] <- rnorm(N, beta.mean[i], sqrt(tau.sq.beta[i]))
}
for (i in 1:p.det) {
  alpha[, i] <- rnorm(N, alpha.mean[i], sqrt(tau.sq.alpha[i]))
}
phi <- runif(N, 3/1, 3/.4)
sigma.sq <- runif(N, 0.3, 3)
sp <- TRUE

dat <- simMsOcc(J.x = J.x, J.y = J.y, n.rep = n.rep, N = N, beta = beta, alpha = alpha,
                phi = phi, sigma.sq = sigma.sq, sp = TRUE, cov.model = 'exponential')

# Number of batches
n.batch <- 40
# Batch length
batch.length <- 25
n.samples <- n.batch * batch.length

y <- dat$y
X <- dat$X
X.p <- dat$X.p
coords <- as.matrix(dat$coords)

# Package all data into a list
occ.covs <- X[, 2, drop = FALSE]
colnames(occ.covs) <- c('occ.cov')
det.covs <- list(det.cov.1 = X.p[, , 2],
                 det.cov.2 = X.p[, , 3])
data.list <- list(y = y,
                  occ.covs = occ.covs,
                  det.covs = det.covs,
                  coords = coords)
# Priors
prior.list <- list(beta.comm.normal = list(mean = 0, var = 2.72),
                   alpha.comm.normal = list(mean = 0, var = 2.72),
                   tau.sq.beta.ig = list(a = 0.1, b = 0.1),
                   tau.sq.alpha.ig = list(a = 0.1, b = 0.1),
                   phi.unif = list(a = 3/1, b = 3/.1),
                   sigma.sq.ig = list(a = 2, b = 2))
```

```
# Initial values
inits.list <- list(alpha.comm = 0,
                   beta.comm = 0,
                   beta = 0,
                   alpha = 0,
                   tau.sq.beta = 1,
                   tau.sq.alpha = 1,
                   phi = 3 / .5,
                   sigma.sq = 2,
                   w = matrix(0, nrow = N, ncol = nrow(X)),
                   z = apply(y, c(1, 2), max, na.rm = TRUE))
# Tuning
tuning.list <- list(phi = 1)

out <- spMsPGOcc(occ.formula = ~ occ.cov,
                 det.formula = ~ det.cov.1 + det.cov.2,
                 data = data.list,
                 inits = inits.list,
                 n.batch = n.batch,
                 batch.length = batch.length,
                 accept.rate = 0.43,
                 priors = prior.list,
                 cov.model = "exponential",
                 tuning = tuning.list,
                 n.omp.threads = 1,
                 verbose = TRUE,
                 NNGP = TRUE,
                 n.neighbors = 5,
                 search.type = 'cb',
                 n.report = 10,
                 n.burn = 500,
                 n.thin = 1)

summary(out, level = 'both')
```

---

spPGOcc *Function for Fitting Single Species Spatial Occupancy Models Using Polya-Gamma Latent Variables*

Description

The function spPGOcc fits single species spatial occupancy models using Polya-Gamma latent variables. Models can be fit using either a full Gaussian process or a Nearest Neighbor Gaussian Process for large data sets.

Usage

```
spPGOcc(occ.formula, det.formula, data, inits, priors,
        tuning, cov.model = "exponential", NNGP = TRUE,
        n.neighbors = 15, search.type = "cb", n.batch,
```

```
batch.length, accept.rate = 0.43,
n.omp.threads = 1, verbose = TRUE, n.report = 100,
n.burn = round(.10 * n.batch * batch.length),
n.thin = 1, k.fold, k.fold.threads = 1,
k.fold.seed = 100, ...)
```

Arguments

occ.formula	a symbolic description of the model to be fit for the occurrence portion of the model using R's model syntax. Only right-hand side of formula is specified. See example below.
det.formula	a symbolic description of the model to be fit for the detection portion of the model using R's model syntax. Only right-hand side of formula is specified. See example below. Random intercepts are allowed using lme4 syntax (Bates et al. 2015).
data	a list containing data necessary for model fitting. Valid tags are y, occ.covs, det.covs, and coords. y is the detection-nondetection data matrix or data frame with first dimension equal to the number of sites (J) and second dimension equal to the maximum number of replicates at a given site. occ.covs is a matrix or data frame containing the variables used in the occupancy portion of the model, with J rows for each column (variable). det.covs is a list of variables included in the detection portion of the model. Each list element is a different detection covariate, which can be site-level or observational-level. Site-level covariates are specified as a vector of length J while observation-level covariates are specified as a matrix or data frame with the number of rows equal to J and number of columns equal to the maximum number of replicates at a given site. coords is a $J \times 2$ matrix of the observation coordinates.
inits	a list with each tag corresponding to a parameter name. Valid tags are z, beta, alpha, sigma.sq, phi, w, nu, and sigma.sq.p. nu is only specified if cov.model = "matern" and sigma.sq.p is only specified if there are random effects in det.formula. The value portion of each tag is the parameter's initial value. See priors description for definition of each parameter name.
priors	a list with each tag corresponding to a parameter name. Valid tags are beta.normal, alpha.normal, phi.unif, sigma.sq.ig, nu.unif, and sigma.sq.p.ig. Occurrence (beta) and detection (alpha) regression coefficients are assumed to follow a normal distribution. The hyperparameters of the normal distribution are passed as a list of length two with the first and second elements corresponding to the mean and variance of the normal distribution, which are each specified as vectors of length equal to the number of coefficients to be estimated or of length one if priors are the same for all coefficients. If not specified, prior means are set to 0 and prior variances set to 2.73. The spatial variance parameter, sigma.sq, is assumed to follow an inverse-Gamma distribution, whereas the spatial decay phi and smoothness nu parameters are assumed to follow Uniform distributions. The hyperparameters of the inverse-Gamma for sigma.sq are passed as a vector of length two, with the first and second elements corresponding to the *shape* and *scale*, respectively. The hyperparameters of the Uniform are also passed as a vector of length two with the first and second elements corresponding to the lower and upper support, respectively. sigma.sq.p

is the random effect variance for any detection random effects and is assumed to follow an inverse-Gamma distribution. For `sigma.sq.p`, the hyperparameters of the inverse-Gamma distribution are passed as a list of length two with the first and second elements corresponding to the shape and scale parameters, respectively, which are each specified as vectors of length equal to the number of random intercepts or of length one if priors are the same for all random effect variances.

`cov.model` a quoted keyword that specifies the covariance function used to model the spatial dependence structure among the observations. Supported covariance model key words are: `"exponential"`, `"matern"`, `"spherical"`, and `"gaussian"`.

`tuning` a list with each tag corresponding to a parameter name. Valid tags are `phi` and `nu`. The value portion of each tag defines the initial variance of the Adaptive sampler. See Roberts and Rosenthal (2009) for details.

`NNGP` if `TRUE`, model is fit with an NNGP. If `FALSE`, a full Gaussian process is used. See Datta et al. (2016) and Finley et al. (2019) for more information.

`n.neighbors` number of neighbors used in the NNGP. Only used if `NNGP = TRUE`. Datta et al. (2016) showed that 15 neighbors is usually sufficient, but that as few as 5 neighbors can be adequate for certain data sets, which can lead to even greater decreases in run time. We recommend starting with 15 neighbors (the default) and if additional gains in computation time are desired, subsequently compare the results with a smaller number of neighbors using WAIC or k-fold cross-validation.

`search.type` a quoted keyword that specifies the type of nearest neighbor search algorithm. Supported method key words are: `"cb"` and `"brute"`. The `"cb"` should generally be much faster. If locations do not have identical coordinate values on the axis used for the nearest neighbor ordering then `"cb"` and `"brute"` should produce identical neighbor sets. However, if there are identical coordinate values on the axis used for nearest neighbor ordering, then `"cb"` and `"brute"` might produce different, but equally valid, neighbor sets, e.g., if data are on a grid.

`n.batch` the number of MCMC batches to run for the Adaptive MCMC sampler. See Roberts and Rosenthal (2009) for details.

`batch.length` the length of each MCMC batch to run for the Adaptive MCMC sampler. See Roberts and Rosenthal (2009) for details.

`accept.rate` target acceptance rate for Adaptive MCMC. Default is 0.43. See Roberts and Rosenthal (2009) for details.

`n.omp.threads` a positive integer indicating the number of threads to use for SMP parallel processing. The package must be compiled for OpenMP support. For most Intel-based machines, we recommend setting `n.omp.threads` up to the number of hyperthreaded cores. Note, `n.omp.threads` > 1 might not work on some systems.

`verbose` if `TRUE`, messages about data preparation, model specification, and progress of the sampler are printed to the screen. Otherwise, no messages are printed.

`n.report` the interval to report Metropolis sampler acceptance and MCMC progress.

`n.burn` the number of samples out of the total `n.batch` * `batch.length` samples to discard as burn-in. By default, the first 10% of samples is discarded.

n.thin	the thinning interval for collection of MCMC samples. The thinning occurs after the n.burn samples are discarded. Default value is set to 1.
k.fold	specifies the number of k folds for cross-validation. If not specified as an argument, then cross-validation is not performed and k.fold.threads and k.fold.seed are ignored. In k-fold cross-validation, the data specified in data is randomly partitioned into k equal sized subsamples. Of the k subsamples, k - 1 subsamples are used to fit the model and the remaining k samples are used for prediction. The cross-validation process is repeated k times (the folds). As a scoring rule, we use the model deviance as described in Hooten and Hobbs (2015). Cross-validation is performed after the full model is fit using all the data. Cross-validation results are reported in the k.fold.deviance object in the return list.
k.fold.threads	number of threads to use for cross-validation. If k.fold.threads > 1 parallel processing is accomplished using the **foreach** and **doParallel** packages. Ignored if k.fold is not specified.
k.fold.seed	seed used to split data set into k.fold parts for k-fold cross-validation. Ignored if k.fold is not specified.
...	currently no additional arguments

Value

An object of class spPGOcc that is a list comprised of:

beta.samples	a coda object of posterior samples for the occurrence regression coefficients.
alpha.samples	a coda object of posterior samples for the detection regression coefficients.
z.samples	a coda object of posterior samples for the latent occurrence values
psi.samples	a coda object of posterior samples for the latent occurrence probability values
theta.samples	a coda object of posterior samples for covariance parameters.
w.samples	a coda object of posterior samples for latent spatial random effects.
sigma.sq.p.samples	a coda object of posterior samples for variances of random intercpets included in the detection portion of the model. Only included if random intercepts are specified in det.formula.
alpha.star.samples	a coda object of posterior samples for the detection random effects. Only included if random intercepts are specified in det.formula.
run.time	execution time reported using proc.time().
k.fold.deviance	soring rule (deviance) from k-fold cross-validation. Only included if k.fold is specified in function call.

The return object will include additional objects used for subsequent prediction and/or model fit evaluation.

Author(s)

Jeffrey W. Doser <doserjef@msu.edu>,
Andrew O. Finley <finleya@msu.edu>

References

Bates, Douglas, Martin Maechler, Ben Bolker, Steve Walker (2015). Fitting Linear Mixed-Effects Models Using lme4. Journal of Statistical Software, 67(1), 1-48. doi: 10.18637/jss.v067.i01.

Datta, A., S. Banerjee, A.O. Finley, and A.E. Gelfand. (2016) Hierarchical Nearest-Neighbor Gaussian process models for large geostatistical datasets. *Journal of the American Statistical Association*, doi: 10.1080/01621459.2015.1044091.

Finley, A.O., A. Datta, B.D. Cook, D.C. Morton, H.E. Andersen, and S. Banerjee. (2019) Efficient algorithms for Bayesian Nearest Neighbor Gaussian Processes. *Journal of Computational and Graphical Statistics*, doi: 10.1080/10618600.2018.1537924.

Finley, A. O., Datta, A., and Banerjee, S. (2020). spNNGP R package for nearest neighbor Gaussian process models. *arXiv* preprint arXiv:2001.09111.

Hooten, M. B., and Hobbs, N. T. (2015). A guide to Bayesian model selection for ecologists. *Ecological Monographs*, 85(1), 3-28.

Hooten, M. B., and Hefley, T. J. (2019). Bringing Bayesian models to life. *CRC Press*.

Polson, N.G., J.G. Scott, and J. Windle. (2013) Bayesian Inference for Logistic Models Using Polya-Gamma Latent Variables. *Journal of the American Statistical Association*, 108:1339-1349.

Roberts, G.O. and Rosenthal J.S. (2009) Examples of adaptive MCMC. *Journal of Computational and Graphical Statistics*, 18(2):349-367.

Examples

```
set.seed(350)
# Simulate Data ------------------------------------------------------------
J.x <- 8
J.y <- 8
J <- J.x * J.y
n.rep <- sample(2:4, J, replace = TRUE)
beta <- c(0.5, -0.15)
p.occ <- length(beta)
alpha <- c(0.7, 0.4, -0.2)
p.det <- length(alpha)
phi <- 3 / .6
sigma.sq <- 2
dat <- simOcc(J.x = J.x, J.y = J.y, n.rep = n.rep, beta = beta, alpha = alpha,
              sigma.sq = sigma.sq, phi = phi, sp = TRUE, cov.model = 'exponential')
y <- dat$y
X <- dat$X
X.p <- dat$X.p
coords <- as.matrix(dat$coords)

# Package all data into a list
occ.covs <- X[, -1, drop = FALSE]
colnames(occ.covs) <- c('occ.cov')
det.covs <- list(det.cov.1 = X.p[, , 2],
 det.cov.2 = X.p[, , 3])
data.list <- list(y = y,
  occ.covs = occ.covs,
  det.covs = det.covs,
```

```
  coords = coords)

# Number of batches
n.batch <- 40
# Batch length
batch.length <- 25
n.iter <- n.batch * batch.length
# Priors
prior.list <- list(beta.normal = list(mean = 0, var = 2.72),
   alpha.normal = list(mean = 0, var = 2.72),
   sigma.sq.ig = c(2, 2),
   phi.unif = c(3/1, 3/.1))
# Initial values
inits.list <- list(alpha = 0, beta = 0,
   phi = 3 / .5,
   sigma.sq = 2,
   w = rep(0, nrow(X)),
   z = apply(y, 1, max, na.rm = TRUE))
# Tuning
tuning.list <- list(phi = 1)

out <- spPGOcc(occ.formula = ~ occ.cov,
       det.formula = ~ det.cov.1 + det.cov.2,
       data = data.list,
       inits = inits.list,
       n.batch = n.batch,
       batch.length = batch.length,
       priors = prior.list,
       cov.model = "exponential",
       tuning = tuning.list,
       NNGP = TRUE,
       n.neighbors = 5,
       search.type = 'cb',
       n.report = 10,
       n.burn = 500)

summary(out)
```

`summary.intPGOcc` *Methods for intPGOcc Object*

Description

Methods for extracting information from fitted single species integrated occupancy (`intPGOcc`) model.

Usage

```
## S3 method for class 'intPGOcc'
summary(object, quantiles = c(0.025, 0.25, 0.5, 0.75, 0.975),
```

```
digits = max(3L, getOption("digits") - 3L), ...)
## S3 method for class 'intPGOcc'
print(x, ...)
```

Arguments

object, x	object of class intPGOcc.
quantiles	for summary, posterior distribution quantiles to compute.
digits	for summary, number of digits to report.
...	currently no additional arguments

Details

A set of standard extractor functions for fitted model objects of class intPGOcc, including methods to the generic functions print and summary.

Value

No return value, called to display summary information of a intPGOcc object.

---

summary.msPGOcc *Methods for msPGOcc Object*

Description

Methods for extracting information from fitted multi-species occupancy (msPGOcc) model.

Usage

```
## S3 method for class 'msPGOcc'
summary(object, level, quantiles = c(0.025, 0.25, 0.5, 0.75, 0.975),
digits = max(3L, getOption("digits") - 3L), ...)
## S3 method for class 'msPGOcc'
print(x, ...)
```

Arguments

object, x	object of class msPGOcc.
level	a quoted keyword that indicates the level to summarize the posterior predictive check. Valid key words are: "community", "species", or "both".
quantiles	for summary, posterior distribution quantiles to compute.
digits	for summary, number of digits to report.
...	currently no additional arguments

Details

A set of standard extractor functions for fitted model objects of class msPGOcc, including methods to the generic functions print and summary.

Value

No return value, called to display summary information of a msPGOcc object.

---

summary.PGOcc *Methods for PGOcc Object*

Description

Methods for extracting information from fitted single species occupancy (PGOcc) model.

Usage

```
## S3 method for class 'PGOcc'
summary(object, quantiles = c(0.025, 0.25, 0.5, 0.75, 0.975),
digits = max(3L, getOption("digits") - 3L), ...)
## S3 method for class 'PGOcc'
print(x, ...)
```

Arguments

object, x	object of class PGOcc.
quantiles	for summary, posterior distribution quantiles to compute.
digits	for summary, number of digits to report.
...	currently no additional arguments

Details

A set of standard extractor functions for fitted model objects of class PGOcc, including methods to the generic functions print and summary.

Value

No return value, called to display summary information of a PGOcc object.

summary.ppcOcc *Methods for ppcOcc Object*

Description

Methods for extracting information from posterior predictive check objects of class ppcOcc.

Usage

```
## S3 method for class 'ppcOcc'
summary(object, level, digits = max(3L, getOption("digits") - 3L), ...)
```

Arguments

object	object of class ppcOcc.
level	a quoted keyword for multispecies models that indicates the level to summarize the posterior predictive check. Valid key words are: "community", "species", or "both".
digits	for summary, number of digits to report.
...	currently no additional arguments

Details

A set of standard extractor functions for fitted posterior predictive check objects of class ppcOcc, including methods to the generic function summary.

Value

No return value, called to display summary information of a ppcOcc object.

summary.spIntPGOcc *Methods for spIntPGOcc Object*

Description

Methods for extracting information from fitted single species integrated occupancy (spIntPGOcc) model.

Usage

```
## S3 method for class 'spIntPGOcc'
summary(object, quantiles = c(0.025, 0.25, 0.5, 0.75, 0.975),
digits = max(3L, getOption("digits") - 3L), ...)
## S3 method for class 'spIntPGOcc'
print(x, ...)
```

Arguments

object, x	object of class spIntPGOcc.
quantiles	for summary, posterior distribution quantiles to compute.
digits	for summary, number of digits to report.
...	currently no additional arguments

Details

A set of standard extractor functions for fitted model objects of class spIntPGOcc, including methods to the generic functions print and summary.

Value

No return value, called to display summary information of a spIntPGOcc object.

---

summary.spMsPGOcc *Methods for spMsPGOcc Object*

Description

Methods for extracting information from fitted multispecies spatial occupancy (spMsPGOcc) model.

Usage

```
## S3 method for class 'spMsPGOcc'
summary(object, level, quantiles = c(0.025, 0.25, 0.5, 0.75, 0.975),
 digits = max(3L, getOption("digits") - 3L), ...)
## S3 method for class 'spMsPGOcc'
print(x, ...)
```

Arguments

object, x	object of class spMsPGOcc.
level	a quoted keyword that indicates the level to summarize the posterior predictive check. Valid key words are: "community", "species", or "both".
quantiles	for summary, posterior distribution quantiles to compute.
digits	for summary, number of digits to report.
...	currently no additional arguments

Details

A set of standard extractor functions for fitted model objects of class spMsPGOcc, including methods to the generic functions print and summary.

Value

No return value, called to display summary information of a spMsPGOcc object.

summary.spPGOcc *Methods for spPGOcc Object*

Description

Methods for extracting information from fitted single species spatial occupancy (spPGOcc) model.

Usage

```
## S3 method for class 'spPGOcc'
summary(object, quantiles = c(0.025, 0.25, 0.5, 0.75, 0.975),
        digits = max(3L, getOption("digits") - 3L), ...)
## S3 method for class 'spPGOcc'
print(x, ...)
```

Arguments

object, x	object of class spPGOcc.
quantiles	for summary, posterior distribution quantiles to compute.
digits	for summary, number of digits to report.
...	currently no additional arguments

Details

A set of standard extractor functions for fitted model objects of class spPGOcc, including methods to the generic functions print and summary.

Value

No return value, called to display summary information of a spPGOcc object.

waicOcc *Compute Widely Applicable Information Criterion for spOccupancy Model Objects*

Description

Function for computing the Widely Applicable Information Criterion (WAIC; Watanabe 2010) for spOccupancy model objects.

Usage

```
waicOcc(object, ...)
```

Arguments

object	an object of class PGOcc, spPGOcc, msPGOcc, spMsPGOcc, intPGOcc, or spIntPGOcc.
...	currently no additional arguments

Details

The effective number of parameters is calculated following the recommendations of Gelman et al. (2014).

Value

When object is of class PGOcc, spPGOcc, msPGOcc, or spMsPGOcc, returns a vector with three elements corresponding to estimates of the expected log pointwise predictive density (elpd), the effective number of parameters (pD), and the WAIC. When object is of class intPGOcc or spIntPGOcc, returns a data frame with columns elpd, pD, and WAIC, with each row corresponding to the estimated values for each data source in the integrated model.

Author(s)

Jeffrey W. Doser <doserjef@msu.edu> Andrew O. Finley <finleya@msu.edu>

References

Watanabe, S. (2010). Asymptotic equivalence of Bayes cross validation and widely applicable information criterion in singular learning theory. *Journal of Machine Learning Research*, 11:3571-3594.

Gelman, A., J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. (2013). Bayesian Data Analysis. 3rd edition. CRC Press, Taylor and Francis Group

Gelman, A., J. Hwang, and A. Vehtari (2014). Understanding predictive information criteria for Bayesian models. *Statistics and Computing*, 24:997-1016.

Examples

```
set.seed(400)
# Simulate Data ------------------------------------------------------------
J.x <- 8
J.y <- 8
J <- J.x * J.y
n.rep <- sample(2:4, J, replace = TRUE)
beta <- c(0.5, -0.15)
p.occ <- length(beta)
alpha <- c(0.7, 0.4)
p.det <- length(alpha)
dat <- simOcc(J.x = J.x, J.y = J.y, n.rep = n.rep, beta = beta, alpha = alpha,
      sp = FALSE)
occ.covs <- dat$X[, 2, drop = FALSE]
colnames(occ.covs) <- c('occ.cov')
det.covs <- list(det.cov = dat$X.p[, , 2])
# Data bundle
```

```
data.list <- list(y = dat$y,
  occ.covs = occ.covs,
  det.covs = det.covs)

# Priors
prior.list <- list(beta.normal = list(mean = rep(0, p.occ),
      var = rep(2.72, p.occ)),
   alpha.normal = list(mean = rep(0, p.det),
               var = rep(2.72, p.det)))
# Initial values
inits.list <- list(alpha = rep(0, p.det),
      beta = rep(0, p.occ),
      z = apply(data.list$y, 1, max, na.rm = TRUE))

n.samples <- 5000
n.report <- 1000

out <- PGOcc(occ.formula = ~ occ.cov,
     det.formula = ~ det.cov,
     data = data.list,
     inits = inits.list,
     n.samples = n.samples,
     priors = prior.list,
     n.omp.threads = 1,
     verbose = TRUE,
     n.report = n.report,
     n.burn = 4000,
     n.thin = 1)

# Calculate WAIC
waicOcc(out)
```

Index