Documentation for R-Package: Richness

A package for asymptotic estimations of species richness from spatial abundance data

Version:	1.1
Depends:	R 4.2.2
Published:	2023-04-27
Last update:	2023-05-25
Authors:	Matt Whalen and Eden Tekwa
Maintainers:	Matt Whalen (mawhal@gmail.com) Eden Tekwa (ewtekwa@gmail.com)
License:	CC BY-NC-SA
URL:	https://github.com/EWTekwa/Richness
Citation:	Tekwa, Whalen, Martone, O'Connor, Theory and application of an improved species richness estimator 2023. Philosophical Transactions B. 10.1098/rstb.2022.0181.2022-0187

Summary:

The R-Package uses spatial abundance or occupancy* data to produce estimated species richness and estimator precision while correcting for observation biases from low occupancies and abundances. The featured estimators include a set from Tekwa et al. 2023: Ω (exact estimator, recommended version), Ω_T (second-order Taylor approximated estimator, useful for quantifying biases from means and variances in occupancy and observed abundance across species), and Ω_o (zeroth-order approximated estimator). Additionally, standard estimates are produced using Chao1, Gamma-Poisson, Chao2, ACE, Jackknife-abundance, Jackknife-incidence, and observed (uncorrected) richness.

A wrapper function *estimateRichness.R* allows the user to provide community data as a matrix, data.frame, list, or array. Lists or arrays can be input to compare multiple communities, such as a time series.

The wrapper function calls two functions *RichnessEsts.R* and *bootstrapRichness.R* that respectively return point estimates and bootstrapped

estimates for estimator precision quantification. These functions take in data matrix with species as columns, spatial units (e.g., transects or quadrats) as rows, and individual counts as values. Only bootstrapped estimates are provided, and it is up to the user to choose how to use these for estimating precision and possibly confidence bounds. For example, precision can be quantified by taking the standard deviation of the bootstrapped estimates.

**note: for occupancy data, all estimators (including Ω) work if the data is first transformed into pseudo-abundance data by multiplying all entries (1 and 0) by 100. This number is large enough to ensure the abundance of observed species does not affect the estimates – only occupancy will.*

Installation instructions:

To download package in R, type:

```
library(devtools)
install_github("EWTekwa/Richness")
library(Richness)
```

To open help files for functions in R, type:

```
?RichnessEsts
?estimate_richness
?bootstrapRichness
```

References for Chao1, Gamma-Poisson, Chao2, ACE, and Jackknife richness estimators:

Chao A. 1984 Nonparametric estimation of the number of classes in a population. Scandinavian J. Stat. 11, 265–270. https://www.jstor.org/stable/4615964

Chiu CH. 2023 A more reliable species richness estimator based on the Gamma–Poisson model. PeerJ 11, e14540. https://doi.org/10.7717/peerj.14540

Chao A. 1987 Estimating the population size
for capture-recapture data with unequal catchability. Biometrics 43, 783–791. https://doi.org/10.2307/2531532

Chao A, Lee SM. 1992 Estimating the number of classes via sample coverage. J. Am. Stat. Assoc. 87, 210–217. 10.1080/01621459.1992.10475194

Burnham KP, Overton WS. 1979 Robust estimation of population size when capture probabilities vary among animals. Ecology 60, 927–936. https://doi.org/10.2307/1936861

Operational instructions (also appears in the R help files):

1. estimateRichness {Richness} R Documentation

Estimate Richness in Community Data

Description

A wrapper function that runs `RichnessEsts` or `bootstrapRichness` using a variety of data input types, including time series community data.

Usage

```
estimateRichness(Community, boot = FALSE, numBoot = 100,
                                   meanStates = FALSE, Apx_detectP_terms = FALSE )
```

Arguments

`Community`	Community data with species as columns and spatial sampling units as rows. Data can take the form of a data frame, matrix, list, or array of 3 dimensions. List and array inputs are useful when calculating richness for community time series or for different communities who richness is to be compared.
`boot`	logical. Should bootstrapped samples be computed?
`numBoot`	The number of bootstrapped samples to return.
`meanStates`	logical. Should mean states of correction terms bootstrapped samples be returned? Only used when `boot = FALSE`
`Apx_detectP_terms`	logical. Should approximated detection probabilities be returned? Only used when `boot = FALSE`

Author(s)

E.W. Tekwa and M.A. Whalen

References

Tekwa, E.W., Whalen, M.A., Martone, P.T. and O'Connor, M.I. (2023) Theory and application of an improved species richness estimator. *Philosophical Transactions of the Royal Society B.* doi: 10.1098/rstb.2022.0181.2022-0187

References for Chao1, Gamma-Poisson, Chao2, ACE, and Jackknife richness estimators:

Chao A. 1984. Nonparametric estimation of the number of classes in a population. *Scandinavian J. Stat.* 11, 265–270. https://www.jstor.org/stable/4615964

Chiu CH. 2023. A more reliable species richness estimator based on the Gamma–Poisson model. *PeerJ.* 11, e14540. doi: 10.7717/peerj.14540

Chao A. 1987 Estimating the population size for capture-recapture data with unequal catchability. *Biometrics.* 43, 783–791. doi: 10.2307/2531532

Chao A, Lee SM. 1992 Estimating the number of classes via sample coverage. *J. Am. Stat. Assoc.* 87, 210–217. doi: 10.1080/01621459.1992.10475194

Burnham KP, Overton WS. 1979. Robust estimation of population size when capture probabilities vary among animals. *Ecology.* 60, 927–936. doi: 10.2307/1936861

See Also

`RichnessEsts`, `bootRichnessEsts`.

Examples

```
## install the package
require(devtools)
install_github("EWTekwa/Richness")
library(Richness)

## Use Barro Colorado Island data from the vegan package
require(vegan)
data("BCI")

## run the function for a single community data frame without bootstrapping
# point estimates only
estimateRichness(BCI)
# with mean states for correction terms and approximated detection
probabilities
estimateRichness(BCI, meanStates = TRUE, Apx_detectP_terms = TRUE)

## run the function for a single community data frame with bootstrapping
estimateRichness(BCI, boot = TRUE, numBoot = 10)

## run the function on a list
# Note: this may not be the most sensible comparison
require(vegan)
data("pyrifos")
week <- gl(11, 12, labels = c(-4, -1, 0.1, 1, 2, 4, 8, 12, 15, 19, 24))
communitylist <- split(pyrifos, f = week)
estimateRichness(communitylist)
# bootstrapping on a list returns a tidy data frame
require(tidyverse)
estimateRichness(communitylist, boot = TRUE, numBoot = 10)
```

[Package *Richness* version 1.1.0 Index]

Richness Estimates with Correction Terms and Detection Probabilities

Description

This function takes spatial abundance data and returns several richness estimates, including three defined by Tekwa et al. 2023. In addition to richness point estimates, the function returns mean states of correction terms and approximated detection probabilities used in Tekwa's estimation method.

Usage

```
RichnessEsts(Community)
```

Arguments

`Community`	Data frame or matrix of community data with species as columns and spatial sampling units as rows.

Value

A list of 3 elements.

`element1`	point estimates for all richness estimators. Estimators include raw richness, Chao1, Gamma-Poisson, Chao2, ACE, Jackknife-abundance, Jackknife-incidence, *Ω*, *Ω{T}*, *Ω{T0}*.
`element2`	mean states of the correction terms for *Ω*, *Ω{T}*, and *Ω{T0}*.
`element3`	approximated detection probabilities for *Ω*, *Ω{T}*, and *Ω{T0}*.

Author(s)

E.W. Tekwa and M.A. Whalen

References

Tekwa, E.W., Whalen, M.A., Martone, P.T. and O'Connor, M.I. (2023) Theory and application of an improved species richness estimator. *Philosophical Transactions of the Royal Society B.* doi: 10.1098/rstb.2022.0181.2022-0187

References for Chao1, Gamma-Poisson, Chao2, ACE, and Jackknife richness estimators:

Chao A. 1984. Nonparametric estimation of the number of classes in a population. *Scandinavian J. Stat.* 11, 265–270. https://www.jstor.org/stable/4615964

Chiu CH. 2023. A more reliable species richness estimator based on the Gamma–Poisson model. *PeerJ.* 11, e14540. doi: 10.7717/peerj.14540

Chao A. 1987 Estimating the population size for capture-recapture data with unequal catchability. *Biometrics.* 43, 783–791. doi: 10.2307/2531532

Chao A, Lee SM. 1992 Estimating the number of classes via sample coverage. *J. Am. Stat. Assoc.* 87, 210–217. doi: 10.1080/01621459.1992.10475194

Burnham KP, Overton WS. 1979. Robust estimation of population size when capture probabilities vary among animals. *Ecology.* 60, 927–936. doi: 10.2307/1936861

See Also

`estimateRichness`, `bootRichnessEsts`.

Examples

```
## install the package
require(devtools)
install_github("EWTekwa/Richness")
library(Richness)

## Use Barro Colorado Island data from the vegan package
require(vegan)
data("BCI")

## run the function, which returns a list
RichnessEsts(BCI)

## isolate the richness estimates
RichnessEsts(BCI)[[1]]
```

[Package *Richness* version 1.1.0 Index]

Bootstrapped Richness Estimates

Description

This function takes spatial abundance data and returns several richness estimates, including two defined by Tekwa et al. 2023. The function uses a bootstrapping routine that can be used to obtain confidence intervals around richness point estimates.

Usage

```
bootstrapRichness(Community, numBoot = 100)
```

Arguments

`Community`	Data frame or matrix of community data with species as columns and spatial sampling units as rows.
`numBoot`	The number of bootstrapped samples to return.

Details

Point estimates are first calculated using `RichnessEsts` and used in the bootstrapping routine.

The bootstrapping routine first resamples rows (sampling units) with replacement, with the number of sampled units the same as the number of rows in the dataset. Next, columns (e.g., species, taxa) are resampled with replacement such that the total number of columns is equal to the number of types calcuated for each richness estimate, generating a data frame for each richness estimate. Then, each richness estimate is calcualted on each resampled data frame, and the bootstap routine repeats until `numBoot` is reached.

Value

A list of 3 elements.

`element1`	point estimates for all richness estimators. Estimators include raw richness, Chao1, Gamma-Poisson, Chao2, ACE, Jackknife-abundance, Jackknife-incidence, *Ω*, *Ω{T}*, *Ω{T0}*.
`element2`	mean states of the correction terms for *Ω*, *Ω{T}*, and *Ω{T0}*.
`element3`	approximated detection probabilities for *Ω*, *Ω{T}*, and *Ω{T0}*.

References

Tekwa, E.W., Whalen, M.A., Martone, P.T. and O'Connor, M.I. (2023) Theory and application of an improved species richness estimator. *Philosophical Transactions of the Royal Society B.* doi: 10.1098/rstb.2022.0181.2022-0187

References for Chao1, Gamma-Poisson, Chao2, ACE, and Jackknife richness estimators:

Chao A. 1984. Nonparametric estimation of the number of classes in a population. *Scandinavian J. Stat.* 11, 265–270. https://www.jstor.org/stable/4615964

Chiu CH. 2023. A more reliable species richness estimator based on the Gamma–Poisson model. *PeerJ.* 11, e14540. doi: 10.7717/peerj.14540

Chao A. 1987 Estimating the population size for capture-recapture data with unequal catchability. *Biometrics.* 43, 783–791. doi: 10.2307/2531532

Chao A, Lee SM. 1992 Estimating the number of classes via sample coverage. *J. Am. Stat. Assoc.* 87, 210–217. doi: 10.1080/01621459.1992.10475194

Burnham KP, Overton WS. 1979. Robust estimation of population size when capture probabilities vary among animals. *Ecology.* 60, 927–936. doi: 10.2307/1936861

See Also

`estimateRichness`, `RichnessEsts`.

Examples

```
## install the package
require(devtools)
install_github("EWTekwa/Richness")
library(Richness)

## Use Barro Colorado Island data from the vegan package
require(vegan)
data("BCI")

## run the function
bootstrapRichness(BCI, numBoot = 10)
```

[Package *Richness* version 1.1.0 Index]

Code:

1. estimate_richness {Richness} R Code

```
# a wrapper function allowing mutliple data input types and
choice of wheter to output point estimes or bootstrapped
estimates
# created by Matt Whalen
# last update 27 Jan 2023
estimateRichness <- function( Community, boot = F, numBoot =
100, meanStates = F, Apx_detectP_terms = F ){
  if( any(class(Community) %in% c("data.frame","matrix")) ){ #
note that mean states and approximated detection probability
terms are only currently available when estimating single
Communities (i.e., no time series or comparison of independent
Communities across space)
    if( boot == T ){
      est = bootstrapRichness( Community, numBoot = numBoot )
    } else if( meanStates == T & Apx_detectP_terms == T ){
        est = RichnessEsts( Community )
      } else if( meanStates == T & Apx_detectP_terms == F ){
        est = RichnessEsts( Community )[1:2]
      } else if( meanStates == F & Apx_detectP_terms == T ){
        est = RichnessEsts( Community )[c(1:3)]
      } else if( meanStates == F & Apx_detectP_terms == F ){
        est = RichnessEsts( Community )[[1]]
      }
  } else if( class(Community) == "list" ){ # each element of the
list should be a Community with taxa as columns and samples as
rows
    if( boot == F ){
      estall = do.call( rbind,lapply( Community, RichnessEsts )
)
      est = do.call( rbind, estall[,1] )
    } else {
      est = do.call( rbind,lapply( Community, bootstrapRichness,
numBoot = numBoot ) )
      estlong = est %>%
        mutate( id = gl( n = length(Community), k = numBoot,
labels = rownames(est) ) ) %>%
        pivot_longer( Richness_raw:JK_i, names_to = "estimate",
values_to = "richness" )
      est = estlong
    }
  } else  if( any(class(Community) == "array" &
length(dim(Community)) > 2) ){ # arrays should be arrange with
taxa as columns, samples as rows, and time/space in the 3rd
dimension
    if( boot == F ){
      estall = do.call( rbind, apply( Community,3, RichnessEsts
) )
      est = do.call( rbind, estall[,1] )
      estlong <- est %>%
        mutate( id = as.numeric(gl( n = dim(Community)[3], k = 1
)) ) %>%
        pivot_longer( Richness_raw:JK_i, names_to = "estimate",
values_to = "richness" )
      est = estlong
    } else {
      est = do.call( rbind,apply( Community, 3,
bootstrapRichness, numBoot = numBoot ) )
      estlong = est %>%
        mutate( id = gl( n = dim(Community)[3], k = numBoot,
labels = rownames(est) ) ) %>%
        pivot_longer( Richness_raw:JK_i, names_to = "estimate",
values_to = "richness" )
      est = estlong
    }
  } else if( !(any(class(Community) %in%
c("data.frame","matrix","list","array"))) ){
    est = c("sorry, this Community data does not appear to be in
a supported format",
            "try coercing the data to a matrix, data.frame,
list, or array with species as columns and spatial sampling
units as rows" )
  }
      return(est)
}
```

2. RichnessEsts {Richness} R Code

```
RichnessEsts <- function( Community ){

# RichnessEsts.R is based on RichnessEsts.m
  # Matt Whalen rewrote matlab script for R - started Mar 27,
2022
# function calculates several estimates of richness:
  # - raw richness
  # - newly proposed method
  # - Chao1
  # - Gamma-Poisson
  # - Chao2
  # - Abundance-based coverage estimator (ACE)
  # - Jackknife abundance estimator
  # - Jackknife incidence estimator
  # - three versions of the estimator introduced in Tekwa et al.
2023
    # - Omega - exact estimator, recommended version
    # - Omega_T - second-order Taylor approximated estimator,
useful for quantifying biases from means and variances in
occupancy and observed abundance across species
    # - Omega_not - zeroth-order approximated estimator

# The new method calculates richness based on spatial Community
data:
  # rows=transects, columns=species, values=individual counts




## write observational process model
# Dix = 1-exp(-nm/P) # local detection probability of species i
at sampled site x
# Dix = Dixo*P
# Di = 1-(1-Dix)^k # detection probability of species i across
all sampled sites x in community s
Di = expression( 1-(1-((1-exp(-nm/P))*P))^k )

## second-order derivatives
# d2Di_dnm2 = D(D( Di, "nm" ), "nm")
# d2Di_dP2  = D(D( Di, "P" ), "P")
# d2Di_dnmP = D(D( Di, "nm" ), "P")

# hard code second-order derivatives above for speed
d2Di_dnm2 = expression( -((1 - ((1 - exp(-nm/P)) * P))^(k - 1) *
                             (k * (exp(-nm/P) * (1/P) * (1/P) *
P)) +
                             (1 - ((1 - exp(-nm/P)) * P))^((k -
1) - 1) *
                             ((k - 1) * (exp(-nm/P) * (1/P) * P))
*
                             (k * (exp(-nm/P) * (1/P) * P))) )
d2Di_dP2  = expression( -((1 - ((1 - exp(-nm/P)) * P))^(k - 1) *
                             (k * (exp(-nm/P) * (nm/P^2) +
                             ((exp(-nm/P) * (nm/P^2) * (nm/P^2) -
exp(-nm/P) *
                             (nm * (2 * P)/(P^2)^2)) * P + exp(-
nm/P) *
                             (nm/P^2)))) + (1 - ((1 - exp(-nm/P))
* P))^((k - 1) - 1) *
                             ((k - 1) * ((1 - exp(-nm/P)) - exp(-
nm/P) * (nm/P^2) * P)) *
                             (k * ((1 - exp(-nm/P)) - exp(-nm/P)
* (nm/P^2) * P))))
d2Di_dnmP = expression( (1 - ((1 - exp(-nm/P)) * P))^(k - 1) *
                             (k * ((exp(-nm/P) * (nm/P^2) * (1/P)
-
                                    exp(-nm/P) * (1/P^2)) * P +
exp(-nm/P) * (1/P))) -
                             (1 - ((1 - exp(-nm/P)) * P))^((k -
1) - 1) *
                             ((k - 1) * ((1 - exp(-nm/P)) - exp(-
nm/P) * (nm/P^2) * P)) *
                             (k * (exp(-nm/P) * (1/P) * P)) )

numSamplingUnit =  nrow(Community) # get number of transects
Richness_raw = sum(colSums(Community)>0) # get raw richness
P_detected = rep(0,ncol(Community)) # array of zeros to record
occupancy for each species
for( species in 1:ncol(Community) ) {
  P_detected[species] = sum(Community[,species] >
0)/numSamplingUnit # occupancy as number of transects occupied
divided by number of transects
}
P_detected[P_detected==0] = NA

# compute on full dataset - means, variances, and covariances
mean_P_detected = mean(P_detected[P_detected>0], na.rm = TRUE )
var_P_detected  = var(P_detected[P_detected>0], na.rm = TRUE )
n_m_detected = colMeans(Community[,colSums(Community)>0])
mean_n_m_detected = mean(n_m_detected)
```

```
var_n_m_detected  =
var(colMeans(Community[,colSums(Community)>0]))
cov_nm_P_detected = cov( data.frame(x =
colMeans(Community[,colSums(Community)>0]), y =
na.omit(P_detected)), use = "complete.obs")
if( length(cov_nm_P_detected[,2])>1 ){
  cov_nm_P_detected = cov_nm_P_detected[1,2]
} else {
  cov_nm_P_detected = 0
}

# calculate singletons, doubletons, and the Chao1 richness
estimator
Community = ceiling(Community)
f1 = sum(colSums(Community) == 1) # number of singleton species
f2 = sum(colSums(Community) == 2) # number of doubleton species
Chao1 = Richness_raw+f1*(f1-1)/(2*(f2+1)) # Chao1 richness
estimator

# compute GP (Gamma-Poisson mixture) estimate (Chiu 2023, peerJ)
f3 = sum(colSums(Community)==3) # number of tripleton species
if( f3 == 0 ){
  f3c = 1
} else {
  f3c = f3
}
if( f1 == 0 ){
  f1c = 1
} else {
  f1c = f1
}
A = 2-(2*f2^2)/(3*f1c*f3c)
if( f2 > 0 ){
  f0Chao1 = f1c^2/(2*f2)
} else {
  f0Chao1 = f1c*(f1c-1)/2
}
if( A < 1 ){
  GP = Richness_raw + f0Chao1*max(c(.5, A))
} else {
  GP = Richness_raw + f0Chao1
}

# compute Chao2 estimate
q1 = sum(colSums(Community >= 1) == 1) # number of species
occurring in one sample only
```

```
q2 = sum(colSums(Community >= 1) == 2) # number of species
occurring in two samples only
m = sum(colSums(Community >= 1)) # total number of samples
Chao2 = Richness_raw + ((m-1)/m)*q1*(q1-1)/(2*(q2+1)) # Chao2
richness estimator

# compute abundance-based coverage estimator (ACE)
S_rare = sum(colSums(Community) <= 10 & colSums(Community) > 0 )
# number of rare species (<=10 individuals)
S_abund = sum(colSums(Community) > 10) # number of abundant
species (>10 individuals)
n_rare = sum(Community[,colSums(Community) <= 10 &
colSums(Community) > 0]) # total number of individuals in rare
species
C_ACE = 1-f1/n_rare # sample coverage estimate
wsumfa = 0
for( a in 1:10 ){
  wsumfa = wsumfa + a*(a-1)*sum(colSums(Community) == a)
}
V2 = max(((S_rare/C_ACE)*wsumfa/(n_rare*(n_rare - 1)) - 1),0) #
coefficient of variation
if( C_ACE > 0 ){
  ACE = S_abund + S_rare/C_ACE + (f1/C_ACE)*V2
} else {
  ACE = Chao1
}

# compute jackknife abundance estimator
JK_a = Richness_raw+2*f1-f2

# compute jackknife incidence estimator
if( m == 1){
  JK_i = Richness_raw
} else {
  JK_i = Richness_raw + (q1*(2*m-3)/m-q2*((m-2)^2)/(m*(m-1)))
}


# compute correction terms for Omega_T
k  = numSamplingUnit
nm = mean_n_m_detected
P  = mean_P_detected
Omega_detectP_terms = c( eval(Di),
  eval(d2Di_dnm2)*var_n_m_detected/2,
  eval(d2Di_dP2)*var_P_detected/2,
  eval(d2Di_dnmP)*cov_nm_P_detected ) # Approximated detection
probability in community
```

```
# check for correction term relative to some threshold
if( sum(Omega_detectP_terms, na.rm = T) > 0.1 ){ # if sum of
correction terms is positive and greater than a threshold
  D_omega = sum(Omega_detectP_terms, na.rm = T) # use full
correction
} else {
  D_omega = Omega_detectP_terms[1] # else, use 0th order
correction
}
if( sum(Omega_detectP_terms, na.rm = T) > 1 ){
  D_omega = 1
}
if( sum(Omega_detectP_terms, na.rm = T) < 0.1 ){
  D_omega = 0.1
}
Omega_taylor   = Richness_raw/D_omega
Omega_taylor_0 = Richness_raw/Omega_detectP_terms[1]


# assign the mean states of correction terms to return with the
function
meanStates = c(mean_n_m_detected,mean_P_detected,
                var_n_m_detected,var_P_detected,
                cov_nm_P_detected )


# compute Omega
nm = n_m_detected
P = na.omit(P_detected)
D_means = eval(Di) # observation process model for each detected
species
D_mean = mean(D_means)
Omega = Richness_raw/D_mean


if( Richness_raw == 0 ){
  Richness_raw = NA
  Richness_taylor = NA
  Richness_taylor_0 = NA
  Richness_omega = NA
  Chao1 = NA
  GP = NA
  Chao2 = NA
  ACE = NA
  JK_a = NA
  JK_i = NA
}



return(list(data.frame(Richness_raw = Richness_raw,
                  Chao1 = Chao1, GP = GP, Chao2 = Chao2,
                  ACE = ACE, JK_a = JK_a, JK_i = JK_i,
                  Omega = Omega, Omega_taylor = Omega_taylor,
Omega_taylor_0 = Omega_taylor_0),
            meanStates = meanStates,
            Omega_detectP_terms = Omega_detectP_terms))
}
```

3. bootstrapRichness {Richness} R Code

```
bootstrapRichness <- function(Community, numBoot = 100){

# bootstrapRichness.R is based on bootRichnessEsts.m
  # Matt Whalen rewrote matlab script for R - started Mar 27,
2022
  # function calculates several estimates of richness:
  # - raw richness
  # - newly proposed method
  # - Chao1
  # - Gamma-Poisson
  # - Chao2
  # - Abundance-based coverage estimator (ACE)
  # - Jackknife abundance estimator
  # - Jackknife incidence estimator
  # - three versions of the estimator introduced in Tekwa et al.
2023
    # - Omega - exact estimator, recommended version
    # - Omega_T - second-order Taylor approximated estimator,
useful for quantifying biases from means and variances in
occupancy and observed abundance across species
    # - Omega_not - zeroth-order approximated estimator

# Point estimates are calculated using script "RichnessEsts.R"
# This script calculates bootstrapped confidence intervals
# the spatial Community data: rows = transects, columns =
species,
#                                     values = individual counts
# Whalen update 9 April 2023
  # - remove Clustering calculations
  # - add estimators Gamma Poisson and edited Omega estimates
# Whalen update 24 May 2023
  # - fixing minor errors in preparation for paper release


# Get point estimates from original dataset
Community = Community[ ,colSums(Community) > 0 ] # remove empty
species columns
# run function to calculate all point estimates - ignore
correction terms states and approximated detection probabilities
pointests = RichnessEsts( Community )[[1]]


# define variance function with normalizatin of n rather than n-
1
varn <- function(x) mean((x-mean(x))^2)
```

```
# store bootstrapped estimates for raw richness and each
estimator
expectedRichness_raw   = rep(0,numBoot)  # raw richness
expectedChao1 = rep(0,numBoot)           # Chao1
expectedGP    = rep(0,numBoot)           # Gamma-Poisson
expectedChao2 = rep(0,numBoot)           # Chao2
expectedACE   = rep(0,numBoot)           # ACE
expectedJK_a  = rep(0,numBoot)           # Jackknife (abundance)
expectedJK_i  = rep(0,numBoot)           # Jackknife (incidence)
expectedOmega = rep(0,numBoot)           # approximate Omega
richness
expectedOmega_taylor   = rep(0,numBoot)  # approximated Omega
richness using Taylor expansion
expectedOmega_taylor_0 = rep(0,numBoot)  # approximated Omega
richness using the first correction term



# write observational process model
Di <- expression( 1-(1-((1-exp(-nm))*P))^k )
## second-order derivatives
# d2Di_dnm2 <- D(D( Di, "nm" ), "nm")
# d2Di_dP2  <- D(D( Di, "P" ), "P")
# d2Di_dnmP  <- D(D( Di, "nm" ), "P")
## hard code second-order derivatives above for speed
d2Di_dnm2 = expression( -((1 - ((1 - exp(-nm/P)) * P))^(k - 1) *
                            (k * (exp(-nm/P) * (1/P) * (1/P) *
P)) +
                            (1 - ((1 - exp(-nm/P)) * P))^((k -
1) - 1) *
                            ((k - 1) * (exp(-nm/P) * (1/P) * P))
*
                            (k * (exp(-nm/P) * (1/P) * P))) )
d2Di_dP2  = expression( -((1 - ((1 - exp(-nm/P)) * P))^(k - 1) *
                            (k * (exp(-nm/P) * (nm/P^2) +
                                   ((exp(-nm/P) * (nm/P^2) *
(nm/P^2) - exp(-nm/P) *
                                       (nm * (2 * P)/(P^2)^2))
* P + exp(-nm/P) *
                                      (nm/P^2)))) + (1 - ((1 -
exp(-nm/P)) * P))^((k - 1) - 1) *
                            ((k - 1) * ((1 - exp(-nm/P)) - exp(-
nm/P) * (nm/P^2) * P)) *
                            (k * ((1 - exp(-nm/P)) - exp(-nm/P)
* (nm/P^2) * P))))
```

```
d2Di_dnmP = expression( (1 - ((1 - exp(-nm/P)) * P))^(k - 1) *
                          (k * ((exp(-nm/P) * (nm/P^2) * (1/P) -
                                   exp(-nm/P) * (1/P^2)) * P +
exp(-nm/P) * (1/P))) -
                          (1 - ((1 - exp(-nm/P)) * P))^((k - 1)
- 1) *
                          ((k - 1) * ((1 - exp(-nm/P)) - exp(-
nm/P) * (nm/P^2) * P)) *
                          (k * (exp(-nm/P) * (1/P) * P)) )



# get the number of sampling units (e.g., transects or quadrats)
numSamplingUnit = nrow(Community)


# bootstrapping - scramble sampling units and resample
for( resample in 1:numBoot ){
  #  scramble sampling units (e.g., transects) first
    sampleSet_init = Community[
sample(1:numSamplingUnit,numSamplingUnit,replace = T), ]

    # using unit-scrambled data, next scramble the species
    sampleSetRaw   = sampleSet_init[
,sample(1:pointests$Richness_raw,pointests$Richness_raw,replace
= T) ]
    sampleSetChao1 = sampleSet_init[
,sample(1:pointests$Richness_raw,round(pointests$Chao1),replace
= T) ]
    sampleSetGP    = sampleSet_init[
,sample(1:pointests$Richness_raw,round(pointests$GP),replace =
T) ]
    sampleSetChao2 = sampleSet_init[
,sample(1:pointests$Richness_raw,round(pointests$Chao2),replace
= T) ]
    sampleSetACE   = sampleSet_init[
,sample(1:pointests$Richness_raw,round(pointests$ACE),replace =
T) ]
    sampleSetJK_a  = sampleSet_init[
,sample(1:pointests$Richness_raw,round(pointests$JK_a),replace =
T) ]
    sampleSetJK_i  = sampleSet_init[
,sample(1:pointests$Richness_raw,round(pointests$JK_i),replace =
T) ]
    sampleSetOmega          = sampleSet_init[
,sample(1:pointests$Richness_raw,round(pointests$Omega),replace
= T) ]
```

```
    sampleSetOmega_taylor   = sampleSet_init[
,sample(1:pointests$Richness_raw,round(pointests$Omega_taylor),r
eplace = T) ]
    sampleSetOmega_taylor_0 = sampleSet_init[
,sample(1:pointests$Richness_raw,round(pointests$Omega_taylor_0)
,replace = T) ]

    # take out empty species columns
    sampleSetRaw   = sampleSetRaw[ , colSums(sampleSetRaw) > 0 ]
    sampleSetChao1 = sampleSetChao1[ , colSums(sampleSetChao1) >
0 ]
    sampleSetGP    = sampleSetGP[ , colSums(sampleSetGP) > 0 ]
    sampleSetChao2 = sampleSetChao2[ , colSums(sampleSetChao2) >
0 ]
    sampleSetACE   = sampleSetACE[ , colSums(sampleSetACE) > 0 ]
    sampleSetJK_a  = sampleSetJK_a[ , colSums(sampleSetJK_a) > 0
]
    sampleSetJK_i  = sampleSetJK_i[ , colSums(sampleSetJK_i) > 0
]
    sampleSetOmega          = sampleSetOmega[ ,
colSums(sampleSetOmega) > 0 ]
    sampleSetOmega_taylor   = sampleSetOmega_taylor[ ,
colSums(sampleSetOmega_taylor) > 0 ]
    sampleSetOmega_taylor_0 = sampleSetOmega_taylor_0[ ,
colSums(sampleSetOmega_taylor_0) > 0 ]

    # compute raw estimate
    Richness_raw_boot = sum(colSums(sampleSetRaw)>0)

    # compute Chao1 estimate
    sampleSetChao1 = ceiling(sampleSetChao1) # convert data to
discrete count
    raw = sum(colSums(sampleSetChao1) > 0)
    f1 = sum(colSums(sampleSetChao1) == 1) # number of singleton
species
    f2 = sum(colSums(sampleSetChao1) == 2) # number of doubleton
species
    Chao1_boot = raw+f1*(f1-1)/(2*(f2+1)) # Chao1 richness
estimator

    # compute Gamma-Poisson mixture estimate (Chiu 2023, peerJ)
    f1 = sum(colSums(sampleSetGP) == 1) # number of singleton
species
    f2 = sum(colSums(sampleSetGP) == 2) # number of doubleton
species
    f3 = sum(colSums(sampleSetGP)==3) # number of tripleton
species
```

```
    if( f3 == 0 ){
      f3c = 1
    } else {
      f3c = f3
    }
    if( f1 == 0 ){
      f1c = 1
    } else {
      f1c = f1
    }
    A = 2-(2*f2^2)/(3*f1c*f3c)
    if( f2 > 0 ){
      f0Chao1 = f1c^2/(2*f2)
    } else {
      f0Chao1 = f1c*(f1c-1)/2
    }
    if( A < 1 ){
      GP_boot = Richness_raw_boot + f0Chao1*max(c(.5, A))
    } else {
      GP_boot = Richness_raw_boot + f0Chao1
    }

    # compute Chao2 estimate
    raw = sum(colSums(sampleSetChao2) > 0)
    q1 = sum(colSums(sampleSetChao2 > 0) == 1) # number of
species occurring in one transect only
    q2 = sum(colSums(sampleSetChao2 > 0) == 2) # number of
species occurring in two transect only
    m = sum(colSums(sampleSetChao2 > 0))
    Chao2_boot = raw + ((m-1)/m)*q1*(q1-1)/(2*(q2+1)) # Chao2
richness estimator

    # compute abundance-based coverage estimator (ACE)
    sampleSetACE = ceiling(sampleSetACE) # convert data to
discrete count
    raw = sum(colSums(sampleSetACE) > 0)
    f1 = sum(colSums(sampleSetACE) == 1) # number of singleton
species
    f2 = sum(colSums(sampleSetACE) == 2) # number of doubleton
species
    S_rare = sum(colSums(sampleSetACE) <= 10) # number of rare
species (< = 10 individuals)
    S_abund = sum(colSums(sampleSetACE) > 10) # number of rare
species (>10 individuals)
    n_rare = sum(sampleSetACE[,colSums(sampleSetACE) <= 10]) #
total number of individuals in rare species
    C_ACE = 1 - f1/n_rare # sample coverage estimate
    wsumfa = 0
    for( a in 1:10 ){
      wsumfa = wsumfa + a*(a-1)*sum(colSums(Community) == a)
    }
    V2 = max(((S_rare/C_ACE)*wsumfa/(n_rare*(n_rare - 1)) -
1),0) # coefficient of variation
    if( C_ACE > 0 ){
      ACE_boot = S_abund + S_rare/C_ACE + (f1/C_ACE)*V2
    } else {
      ACE_boot = Chao1
    }

    # compute jackknife abundance estimator
    sampleSetACE = ceiling(sampleSetACE) # convert data to
discrete count
    raw = sum(colSums(sampleSetJK_a) > 0)
    f1 = sum(colSums(sampleSetJK_a) == 1) # number of singleton
species
    f2 = sum(colSums(sampleSetJK_a) == 2) # number of doubleton
species
    JK_a_boot = raw + 2*f1-f2

    # compute jackknife incidence estimator
    raw = sum(colSums(sampleSetJK_i) > 0)
    q1 = sum(colSums(sampleSetJK_i > 0) == 1) # number of
species occurring in one transect only
    q2 = sum(colSums(sampleSetJK_i > 0) == 2) # number of
species occurring in two transect only
    m = sum(colSums(sampleSetJK_i > 0))
    if( m == 1){
      JK_i_boot = rawli
    } else {
      JK_i_boot = raw + (q1*(2*m-3)/m-q2*((m-2)^2)/(m*(m-1)))
    }

    # Omega_taylor
    # compute correction terms for proposed approximation method
    raw = sum(colSums(sampleSetOmega_taylor) > 0)
    P_detected = rep(0,length(raw)) # array of zeros to record
occupancy for each species
    for( species in 1:ncol(sampleSetOmega_taylor) ) {
      P_detected[species] = sum(sampleSetOmega_taylor[,species]
> 0)/numSamplingUnit # occupancy as number of transects occupied
divided by number of transects
    }
    P_detected[P_detected==0] = NA
    # compute on full dataset - means, variances, and
covariances
    mean_P_detected = mean(P_detected[P_detected>0], na.rm =
TRUE )
    var_P_detected  = var(P_detected[P_detected>0], na.rm = TRUE
)
    n_m_detected =
colMeans(sampleSetOmega_taylor[,colSums(sampleSetOmega_taylor)>0
])
    mean_n_m_detected = mean(n_m_detected)
    var_n_m_detected  =
var(colMeans(sampleSetOmega_taylor[,colSums(sampleSetOmega_taylo
r)>0]))
    cov_nm_P_detected = cov( data.frame(x =
colMeans(sampleSetOmega_taylor[,colSums(sampleSetOmega_taylor)>0
]), y = na.omit(P_detected)), use = "complete.obs")
    if( length(cov_nm_P_detected[,2])>1 ){
      cov_nm_P_detected = cov_nm_P_detected[1,2]
    } else {
      cov_nm_P_detected = 0
    }

    k = numSamplingUnit
    nm = mean_n_m_detected
    P = mean_P_detected
    # add "eval" before "(" below if using ke in place of k
    Taylor_detectP_terms = (c( eval(Di),
                            eval(d2Di_dnm2)*var_n_m_detected/2,
                            eval(d2Di_dP2)*var_P_detected/2,
                            eval(d2Di_dnmP)*cov_nm_P_detected ))
# Approximated detection probability in community

    # check for correction term relative to some threshold
    if( sum(Taylor_detectP_terms, na.rm = T) > 0.1 ){ # if sum
of correction terms is positive and greater than a threshold
      D_omega = sum(Taylor_detectP_terms, na.rm = T) # use full
correction
    } else {
      D_omega = Taylor_detectP_terms[1] # else, use 0th order
correction
    }
    if( sum(Taylor_detectP_terms, na.rm = T) > 1 ){
      D_omega = 1
    }
    Omega_taylor_boot = raw/D_omega

    # Omega_talor_0 using first detection term
    raw = sum(colSums(sampleSetOmega_taylor_0) > 0)
    P_detected = rep(0,length(raw)) # array of zeros to record
occupancy for each species
    for( species in 1:ncol(sampleSetOmega_taylor_0) ) {
      P_detected[species] =
sum(sampleSetOmega_taylor_0[,species] > 0)/numSamplingUnit #
occupancy as number of transects occupied divided by number of
transects
    }
    P_detected[P_detected==0] = NA
    # compute on full dataset - means, variances, and
covariances
    mean_P_detected = mean(P_detected[P_detected>0], na.rm =
TRUE )
    n_m_detected =
colMeans(sampleSetOmega_taylor_0[,colSums(sampleSetOmega_taylor_
0)>0])
    mean_n_m_detected = mean(n_m_detected)

    k = numSamplingUnit
    nm = mean_n_m_detected
    P = mean_P_detected
    Omega_taylor_0_boot = raw/eval(Di)

    # compute Omega
    nm = n_m_detected
    P = na.omit(P_detected)
    D_means = eval(Di)
    D_mean = mean(D_means, na.rm = T)
    Omega_boot = raw/D_mean

    expectedRichness_raw[resample]   = Richness_raw_boot
    expectedChao1[resample] = Chao1_boot
    expectedGP[resample]    = GP_boot
    expectedChao2[resample] = Chao2_boot
    expectedACE[resample]   = ACE_boot
    expectedJK_a[resample] = JK_a_boot
    expectedJK_i[resample] = JK_i_boot
    expectedOmega[resample] = Omega_boot
    expectedOmega_taylor[resample]   = Omega_taylor_boot
    expectedOmega_taylor_0[resample] = Omega_taylor_0_boot
}



return(data.frame(Richness_raw = expectedRichness_raw,
            Omega = expectedOmega,
            Omega_T = expectedOmega_taylor,
            Omega_T0 = expectedOmega_taylor_0,
            Chao1 = expectedChao1, Chao2 = expectedChao2,
            ACE = expectedACE, JK_a = expectedJK_a, JK_i =
expectedJK_i))
}
```