Package 'SongEvo'

January 20, 2025

Title An Individual-Based Model of Bird Song Evolution

Version 1.0.0

BugReports https://github.com/raydanner/SongEvo/issues

Description Simulates the cultural evolution of quantitative traits of bird song. 'SongEvo' is an individual- (agent-) based model. 'SongEvo' is spatially-explicit and can be parameterized with, and tested against, measured song data. Functions are available for model implementation, sensitivity analyses, parameter optimization, model validation, and hypothesis testing.

Depends R (>= 3.1.0)

Imports boot, geosphere, lattice, sp

Suggests reshape2, Hmisc, knitr, rmarkdown

License GPL-3

LazyData true

VignetteBuilder knitr

Encoding UTF-8

RoxygenNote 6.1.1

NeedsCompilation no

Author Raymond Danner [aut, cre],
Elizabeth Derryberry [aut],
Graham Derryberry [aut],
Julie Danner [aut],
David Luther [aut]

Maintainer Raymond Danner <dannerR@uncw.edu>

Repository CRAN

Date/Publication 2019-03-05 19:10:09 UTC

Contents

`glo.parms` *Default model parameters*

Description

Default model parameters for WCSP male trill bandwidth in 1969

Usage

```
glo.parms
```

Format

A data frame with 89 rows and 3 variables:

learning.error.d Drift in trill bw due to learning error

learning.error.sd Dispersal in trill bw due to learning error

global.parms$learning.error.d NULL

n.territories Number of available territories

mortality.a Mortality a

mortality.j Mortality J

lifespan Mean WCSP lifespan

phys.lim.min Minimum possible trill bandwidth

phys.lim.max Maximum possible trill bandwidth

male.fledge.n.mean Mean number of males fledged

male.fledge.n.sd Std deviation of the number of males fledged

disp.age Age at which males disperse

disp.distance.mean Mean dispersal distance

disp.distance.sd Std deviation of dispersal distance

terr.turnover Relative number males who lose their territory

male.fledge.n vector of length n.territories with number of males fledged for each territory

h.test *Test hypotheses about song evolution*

Description

Test if cultural traits evolve through specific mechanisms (e.g. drift or selection).

Usage

```
h.test(summary.results, ts, target.data)
```

Arguments

summary.results	The summary.results array (i.e. a multi-dimensional table) from SongEvo(), which includes population summary values for each time step (dimension 1) in each iteration (dimension 2) of the model. Population summary values are contained in five additional dimensions: population size for each time step of each iteration ("sample.n"), the population mean and variance of the song feature studied ("trait.pop.mean" and "trait.pop.variance"), with associated lower ("lci") and upper ("uci") confidence intervals.
ts	The timestep ("ts") at which to compare simulated trait values to empirical trait values ("empir.trait").
target.data	Trait values from the test population to compare to simulated results. May be measured (i.e. empirical) or hypothetical.

Value

a list with two measures of accuracy: 1. The proportion of observed points that fall within the confidence intervals of the simulated data and the residuals between simulated and observed population trait means; 2. Precision is measured as the residuals between simulated and observed population trait variances.

See Also

SongEvo, par.sens, par.opt, mod.val

Examples

```
### See vignette for an example that uses all functions in SongEvo.

#Prepare initial song data for Bear Valley.
data("song.data")
data("glo.parms")
years=2005-1969
iteration=5
timestep=1
n.territories <- glo.parms$n.territories
```

```
starting.trait <- subset(song.data, Population=="Bear Valley" & Year==1969)$Trill.FBW
starting.trait2 <- c(starting.trait, rnorm(n.territories-length(starting.trait),
                                    mean=mean(starting.trait), sd=sd(starting.trait)))
init.inds <- data.frame(id = seq(1:n.territories), age = 2, trait = starting.trait2)
init.inds$x1 <-  round(runif(n.territories, min=-122.481858, max=-122.447270), digits=8)
init.inds$y1 <-  round(runif(n.territories, min=37.787768, max=37.805645), digits=8)

#Specify and call SongEvo() with test data

SongEvo3 <- with(glo.parms,SongEvo(init.inds = init.inds,
                    iteration = iteration,
                    steps = years,
                    timestep = timestep,
                    n.territories = n.territories,
                    terr.turnover = terr.turnover,
                    learning.method = "integrate",
                    integrate.dist = 50,
                    learning.error.d = learning.error.d,
                    learning.error.sd = learning.error.sd,
                    mortality.a = mortality.a,
                    mortality.j = mortality.j,
                    lifespan = NA,
                    phys.lim.min = phys.lim.min,
                    phys.lim.max = phys.lim.max,
                    male.fledge.n.mean = male.fledge.n.mean,
                    male.fledge.n.sd = male.fledge.n.sd,
                    male.fledge.n = male.fledge.n,
                    disp.age = disp.age,
                    disp.distance.mean = disp.distance.mean,
                    disp.distance.sd = disp.distance.sd,
                    mate.comp = FALSE,
                    prin = FALSE,
                    all = FALSE))

#Specify and call `h.test()`
ts=years
target.data <- subset(song.data, Population=="Bear Valley" & Year==2005)$Trill.FBW
h.test1 <- h.test(summary.results=SongEvo3$summary.results, ts=ts, target.data=target.data)

# The output data list includes two measures of accuracy: the proportion of
# observed points that fall within the confidence intervals of the simulated
# data and the residuals between simulated and observed population trait means.
# Precision is measured as the residuals between simulated and observed
# population trait variances.

# Eighty percent of the observed data fell within the central 95% of the
# simulated values, providing support for the hypothesis that cultural drift as
# described in this model is sufficient to describe the evolution of trill
# frequency bandwidth in this population.
h.test1

## Not run:
#Plot simulated data in relation to measured data.
```

```
#Plot
plot(SongEvo3$summary.results[1, , "trait.pop.mean"],
     xlab="Year", ylab="Bandwidth (Hz)", xaxt="n", type="n",
   xlim=c(-0.5, 35.5), ylim=range(SongEvo3$summary.results[, , "trait.pop.mean"], na.rm=TRUE))
for(p in 1:iteration){
lines(SongEvo3$summary.results[p, , "trait.pop.mean"], col="light gray")
}
freq.mean <- apply(SongEvo3$summary.results[, , "trait.pop.mean"], 2, mean, na.rm=TRUE)
lines(freq.mean, col="blue")
axis(side=1, at=seq(0, 35, by=5), labels=seq(1970, 2005, by=5))#, tcl=-0.25, mgp=c(2,0.5,0))
#Plot 95% quantiles (which are similar to credible intervals)
quant.means <- apply (SongEvo3$summary.results[, , "trait.pop.mean"], MARGIN=2,
                      quantile, probs=c(0.95, 0.05), R=600, na.rm=TRUE)
lines(quant.means[1,], col="blue", lty=2)
lines(quant.means[2,], col="blue", lty=2)
 #plot original song values
library("boot")
sample.mean <- function(d, x) {
mean(d[x])
}
boot_hist <- boot(starting.trait, statistic=sample.mean, R=100)#, strata=mn.res$iteration)

ci.hist <- boot.ci(boot_hist, conf=0.95, type="basic")
low <- ci.hist$basic[4]
high <- ci.hist$basic[5]
points(0, mean(starting.trait), pch=20, cex=0.6, col="black")
library("Hmisc")
errbar(x=0, y=mean(starting.trait), high, low, add=TRUE)
 #plot current song values
points(rep(ts, length(target.data)), target.data)
library("boot")
sample.mean <- function(d, x) {
mean(d[x])
}
boot_curr <- boot(target.data, statistic=sample.mean, R=100)#, strata=mn.res$iteration)
ci.curr <- boot.ci(boot_curr, conf=0.95, type="basic")
low <- ci.curr$basic[4]
high <- ci.curr$basic[5]
points(years, mean(target.data), pch=20, cex=0.6, col="black")
library("Hmisc")
errbar(x=years, y=mean(target.data), high, low, add=TRUE)
 #text and arrows
text(x=11, y=2850, labels="Historical songs", pos=1)
arrows(x0=5, y0=2750, x1=0.4, y1=mean(starting.trait), length=0.1)
text(x=25, y=2900, labels="Current songs", pos=1)
arrows(x0=25, y0=2920, x1=years, y1=mean(target.data), length=0.1)

## End(Not run)
```

`mod.val` *Model validation*

Description

This function allows users to assess the validity of the specified model by testing model performance with a population different from the population used to build the model. The user first runs SongEvo with initial trait values from the validation population.

Usage

```
mod.val(summary.results, ts, target.data)
```

Arguments

summary.results	The summary.results array (i.e. a multi-dimensional table) from SongEvo(), which includes population summary values for each time step (dimension 1) in each iteration (dimension 2) of the model. Population summary values are contained in five additional dimensions: population size for each time step of each iteration ("sample.n"), the population mean and variance of the song feature studied ("trait.pop.mean" and "trait.pop.variance"), with associated lower ("lci") and upper ("uci") confidence intervals.
ts	The timestep ("ts") at which to compare simulated trait values to empirical trait values ("target.data").
target.data	Trait values from the validation population to compare to simulated results. May be measured (i.e. empirical) or hypothetical.

Value

Three measurements of accuracy: i) the mean of absolute residuals of the predicted population mean values in relation to observed values (smaller absolute residuals indicate a more accurate model), ii) the difference between the bootstrapped mean of predicted population means and the mean of the observed values, and iii) the proportion of simulated population trait means that fall within confidence intervals of the observed data (a higher proportion indicates greater accuracy). Precision is measured with the residuals of the predicted population variance to the variance of observed values (smaller residuals indicate a more precise model). Users specify the timestep ("ts") at which to compare simulated trait values to empirical trait values ("empir.trait").

See Also

SongEvo, par.sens, par.opt, h.test

Examples

```
### See vignette for an example that uses all functions in SongEvo.

#Parameterize SongEvo with initial song data from Schooner Bay, CA in 1969, and
#then compare simulated data to target (i.e. observed) data in 2005.

data("song.data")
data("glo.parms")
```

```
list2env(glo.parms, globalenv())
#Prepare initial song data for Schooner Bay.
starting.trait <- subset(song.data, Population=="Schooner" & Year==1969)$Trill.FBW
starting.trait2 <- c(starting.trait, rnorm(n.territories-length(starting.trait),
                                           mean=mean(starting.trait),
                                           sd=sd(starting.trait)))
init.inds <- data.frame(id = seq(1:n.territories), age = 2, trait = starting.trait2)
init.inds$x1 <-  round(runif(n.territories, min=-122.481858, max=-122.447270), digits=8)
init.inds$y1 <-  round(runif(n.territories, min=37.787768, max=37.805645), digits=8)

#Specify and call SongEvo() with validation data
iteration <- 5
years <- 36
timestep <- 1
terr.turnover <- 0.5
SongEvo2 <- SongEvo(init.inds = init.inds,
                    iteration = iteration,
                    steps = years,
                    timestep = timestep,
                    n.territories = n.territories,
                    terr.turnover = terr.turnover,
                    learning.method = "integrate",
                    integrate.dist = 50,
                    learning.error.d = learning.error.d,
                    learning.error.sd = learning.error.sd,
                    mortality.a = mortality.a,
                    mortality.j = mortality.j,
                    lifespan = NA,
                    phys.lim.min = phys.lim.min,
                    phys.lim.max = phys.lim.max,
                    male.fledge.n.mean = male.fledge.n.mean,
                    male.fledge.n.sd = male.fledge.n.sd,
                    male.fledge.n = male.fledge.n,
                    disp.age = disp.age,
                    disp.distance.mean = disp.distance.mean,
                    disp.distance.sd = disp.distance.sd,
                    mate.comp = TRUE,
                    prin = TRUE,
                    all=FALSE)

#Specify and call mod.val
ts <- 36
target.data <- subset(song.data, Population=="Schooner" & Year==2005)$Trill.FBW
mod.val1 <- mod.val(summary.results=SongEvo2$summary.results, ts=ts, target.data=target.data)


#Plot results from `mod.val()`
plot(SongEvo2$summary.results[1, , "trait.pop.mean"],
     xlab="Year", ylab="Bandwidth (Hz)", xaxt="n", type="n",
   xlim=c(-0.5, 36.5), ylim=range(SongEvo2$summary.results[, , "trait.pop.mean"], na.rm=TRUE))
for(p in 1:iteration){
lines(SongEvo2$summary.results[p, , "trait.pop.mean"], col="light gray")
}
```

```
freq.mean <- apply(SongEvo2$summary.results[, , "trait.pop.mean"], 2, mean, na.rm=TRUE)
lines(freq.mean, col="blue")
axis(side=1, at=seq(0, 35, by=5), labels=seq(1970, 2005, by=5))#, tcl=-0.25, mgp=c(2,0.5,0))
#Plot 95% quantiles
quant.means <- apply (SongEvo2$summary.results[, , "trait.pop.mean"], MARGIN=2,
                      quantile, probs=c(0.95, 0.05), R=600, na.rm=TRUE)
lines(quant.means[1,], col="blue", lty=2)
lines(quant.means[2,], col="blue", lty=2)
#plot mean and CI for historic songs.
 #plot original song values
library("boot")
sample.mean <- function(d, x) {
mean(d[x])
}
boot_hist <- boot(starting.trait, statistic=sample.mean, R=100)
ci.hist <- boot.ci(boot_hist, conf=0.95, type="basic")
low <- ci.hist$basic[4]
high <- ci.hist$basic[5]
points(0, mean(starting.trait), pch=20, cex=0.6, col="black")
library("Hmisc")
errbar(x=0, y=mean(starting.trait), high, low, add=TRUE)
 #text and arrows
text(x=5, y=2720, labels="Historical songs", pos=1)
arrows(x0=5, y0=2750, x1=0.4, y1=mean(starting.trait), length=0.1)
 #plot current song values
library("boot")
sample.mean <- function(d, x) {
mean(d[x])
}
boot_curr <- boot(target.data, statistic=sample.mean, R=100)
ci.curr <- boot.ci(boot_curr, conf=0.95, type="basic")
low <- ci.curr$basic[4]
high <- ci.curr$basic[5]
points(years, mean(target.data), pch=20, cex=0.6, col="black")
library("Hmisc")
errbar(x=years, y=mean(target.data), high, low, add=TRUE)
 #text and arrows
text(x=25, y=3100, labels="Current songs", pos=3)
arrows(x0=25, y0=3300, x1=36, y1=mean(target.data), length=0.1)
```

`par.opt` *Parameter optimization*

Description

This function follows par.sens to help users optimize values for imperfectly known parameters for SongEvo. The goals are to maximize accuracy and precision of model prediction.

Usage

```
par.opt(sens.results, ts, target.data, par.range)
```

Arguments

sens.results	The sens.results array from par.sens(), which includes summary.results from SongEvo() for a range of parameter values. summary.results from SongEvo() includes population summary values for each time step (dimension 1) in each iteration (dimension 2) of the model. Population summary values are contained in five additional dimensions: population size for each time step of each iteration ("sample.n"), the population mean and variance of the song feature studied ("trait.pop.mean" and "trait.pop.variance"), with associated lower ("lci") and upper ("uci") confidence intervals.
ts	The timestep ("ts") at which to compare simulated trait values to target trait values ("target.data").
target.data	A set of trait values against which to compare simulated values. target.data may be measured (e.g. from a training population) or hypothetical.
par.range	Range of parameter values over which to optimize.

Value

Three measurements of accuracy and one measure of precision. Accuracy is quantified by three different approaches: i) the mean of absolute residuals of the predicted population mean values in relation to observed values (smaller absolute residuals indicate a more accurate model), ii) the difference between the bootstrapped mean of predicted population means and the mean of the observed values, and iii) the proportion of simulated population trait means that fall within confidence intervals of the observed data (a higher proportion indicates greater accuracy). Precision is measured with the residuals of the predicted population variance to the variance of observed values (smaller residuals indicate a more precise model).

See Also

SongEvo, par.sens, mod.val, h.test

Examples

```
### See vignette for an example that uses all functions in SongEvo.

#### Specify and call `par.sens()`

# Here we test the sensitivity of the Acquire a Territory submodel to variation
# in territory turnover rates, ranging from 0.8-1.2 times the published rate
# (40-60% of territories turned over). The call for the par.sens function has a
# format similar to SongEvo. The user specifies the parameter to test and the
# range of values for that parameter. The function currently allows examination
# of only one parameter at a time and requires at least two iterations.
parm <- "terr.turnover"
par.range = seq(from=0.45, to=0.55, by=0.05)
sens.results <- NULL
```

```
data("song.data")
data("glo.parms")
years=2005-1969
iteration=5
timestep=1
n.territories <- glo.parms$n.territories
starting.trait <- subset(song.data, Population=="PRBO" & Year==1969)$Trill.FBW
starting.trait2 <- c(starting.trait, rnorm(n.territories-length(starting.trait),
                                    mean=mean(starting.trait), sd=sd(starting.trait)))
init.inds <- data.frame(id = seq(1:n.territories), age = 2, trait = starting.trait2)
init.inds$x1 <-  round(runif(n.territories, min=-122.481858, max=-122.447270), digits=8)
init.inds$y1 <-  round(runif(n.territories, min=37.787768, max=37.805645), digits=8)

# Now we call the par.sens function with our specifications.
extra_parms <- list(init.inds = init.inds,
                    timestep = 1,
                    n.territories = nrow(init.inds),
                    learning.method = "integrate",
                    integrate.dist = 0.1,
                    lifespan = NA,
                    terr.turnover = 0.5,
                    mate.comp = FALSE,
                    prin = FALSE,
                    all = TRUE)
global_parms_key <- which(!names(glo.parms) %in% names(extra_parms))
extra_parms[names(glo.parms[global_parms_key])]=glo.parms[global_parms_key]
par.sens1 <- par.sens(parm = parm, par.range = par.range,
                      iteration = iteration, steps = years, mate.comp = FALSE,
                      fixed_parms=extra_parms[names(extra_parms)!=parm], all = TRUE)

#### Prepare current song values
target.data <- subset(song.data, Population=="PRBO" & Year==2005)$Trill.FBW

#### Specify and call `par.opt()`

# Users specify the timestep ("ts") at which to compare simulated trait values
# to target trait data ("target.data") and save the results in an object (called
# `par.opt1` here).
ts <- years
par.opt1 <- par.opt(sens.results=par.sens1$sens.results, ts=ts,
                    target.data=target.data, par.range=par.range)

# Examine results objects (residuals and target match).
par.opt1$Residuals
par.opt1$Target.match



#### Plot results of `par.opt()`
#### Accuracy

  #1. Difference in means.
plot(par.range, par.opt1$Target.match[,1], type="l",
```

```
     xlab="Parameter range", ylab="Difference in means (Hz)")

  #2. Plot proportion contained.
plot(par.range, par.opt1$Prop.contained, type="l",
     xlab="Parameter range", ylab="Proportion contained")

  #3. Calculate and plot mean and quantiles of residuals of mean trait values.
res.mean.means <- apply(par.opt1$Residuals[, , 1], MARGIN=1,
                        mean, na.rm=TRUE)
res.mean.quants <- apply (par.opt1$Residuals[, , 1], MARGIN=1,
                          quantile, probs=c(0.975, 0.025), R=600, na.rm=TRUE)
plot(par.range, res.mean.means, col="orange",
     ylim=range(par.opt1$Residuals[,,1], na.rm=TRUE),
     type="b",
     xlab="Parameter value (territory turnover rate)",
     ylab="Residual of trait mean (trill bandwidth, Hz)")
points(par.range, res.mean.quants[1,], col="orange")
points(par.range, res.mean.quants[2,], col="orange")
lines(par.range, res.mean.quants[1,], col="orange", lty=2)
lines(par.range, res.mean.quants[2,], col="orange", lty=2)

#### Precision
#Calculate and plot mean and quantiles of residuals of variance of trait values
res.var.mean <- apply(par.opt1$Residuals[, , 2], MARGIN=1,
                      mean, na.rm=TRUE)
res.var.quants <- apply (par.opt1$Residuals[, , 2], MARGIN=1,
                         quantile, probs=c(0.975, 0.025), R=600, na.rm=TRUE)
plot(par.range, res.var.mean, col="purple",
     ylim=range(par.opt1$Residuals[,,2], na.rm=TRUE),
     type="b",
     xlab="Parameter value (territory turnover rate)",
     ylab="Residual of trait variance (trill bandwidth, Hz)")
points(par.range, res.var.quants[1,], col="purple")
points(par.range, res.var.quants[2,], col="purple")
lines(par.range, res.var.quants[1,], col="purple", lty=2)
lines(par.range, res.var.quants[2,], col="purple", lty=2)

#### Visual inspection of accuracy and precision: plot trait values for range of parameters
par(mfcol=c(3,2),
    mar=c(4.1, 4.1, 1, 1),
    cex=1.2)
for(i in 1:length(par.range)){
plot(par.sens1$sens.results[ , , "trait.pop.mean", ],
     xlab="Year", ylab="Bandwidth (Hz)",
     xaxt="n", type="n",
   xlim=c(-0.5, years), ylim=range(par.sens1$sens.results[ , , "trait.pop.mean", ], na.rm=TRUE))
for(p in 1:iteration){
lines(par.sens1$sens.results[p, , "trait.pop.mean", i], col="light gray")
}
freq.mean <- apply(par.sens1$sens.results[, , "trait.pop.mean", i], 2, mean, na.rm=TRUE)
lines(freq.mean, col="blue")
axis(side=1, at=seq(0, 35, by=5), labels=seq(1970, 2005, by=5))#, tcl=-0.25, mgp=c(2,0.5,0))
#Plot 95% quantiles
```

```
quant.means <- apply (par.sens1$sens.results[, , "trait.pop.mean", i], MARGIN=2,
                      quantile, probs=c(0.95, 0.05), R=600, na.rm=TRUE)
lines(quant.means[1,], col="blue", lty=2)
lines(quant.means[2,], col="blue", lty=2)
#plot mean and CI for historic songs.
 #plot original song values
library("boot")
sample.mean <- function(d, x) {
mean(d[x])
}
boot_hist <- boot(starting.trait, statistic=sample.mean, R=100)#, strata=mn.res$iteration)

ci.hist <- boot.ci(boot_hist, conf=0.95, type="basic")
low <- ci.hist$basic[4]
high <- ci.hist$basic[5]
points(0, mean(starting.trait), pch=20, cex=0.6, col="black")
library("Hmisc")
errbar(x=0, y=mean(starting.trait), high, low, add=TRUE)

 #plot current song values
boot_curr <- boot(target.data, statistic=sample.mean, R=100)#, strata=mn.res$iteration)
ci.curr <- boot.ci(boot_curr, conf=0.95, type="basic")
low <- ci.curr$basic[4]
high <- ci.curr$basic[5]
points(years, mean(target.data), pch=20, cex=0.6, col="black")
errbar(x=years, y=mean(target.data), high, low, add=TRUE)
  #plot panel title
text(x=3, y=max(par.sens1$sens.results[ , , "trait.pop.mean", ], na.rm=TRUE)-100,
     labels=paste("Par = ", par.range[i], sep=""))
}
```

`par.sens` *Parameter sensitivity*

Description

This function allows testing the sensitivity of SongEvo to different parameter values.

Usage

```
par.sens(parm, par.range, iteration, steps, mate.comp, fixed_parms, all)
```

Arguments

`parm`	The parameter for which to test sensitivity over one or more values.
`par.range`	List of ranges of parameter values over which to test sensitivity.
`iteration`	The number of iterations that the model will run.
`steps`	The number of steps (e.g. years) per iteration.

mate.comp	Female preference for mates. Currently specified as "Yes" or "No".
fixed_parms	Named boolean vector identifying which parameters to keep fixed.
all	Save data for all individuals? Options are TRUE or FALSE.
	The function currently allows examination of only one parameter at a time and requires at least two iterations.

Value

array named sens.results. The sens.results array from par.sens(), which includes summary.results from SongEvo() for a range of parameter values. summary.results from SongEvo() includes population summary values for each time step (dimension 1) in each iteration (dimension 2) of the model. Population summary values are contained in five additional dimensions: population size for each time step of each iteration ("sample.n"), the population mean and variance of the song feature studied ("trait.pop.mean" and "trait.pop.variance"), with associated lower ("lci") and upper ("uci") confidence intervals.

See Also

SongEvo, par.opt, mod.val, h.test

Examples

```
### See vignette for an example that uses all functions in SongEvo.

#### Specify and call `par.sens()`

# Here we test the sensitivity of the Acquire a Territory submodel to variation
# in territory turnover rates, ranging from 0.8-1.2 times the published rate
# (40-60% of territories turned over). The call for the par.sens function has a
# format similar to SongEvo. The user specifies the parameter to test and the
# range of values for that parameter. The function currently allows examination
# of only one parameter at a time and requires at least two iterations.
parm <- "terr.turnover"
par.range = seq(from=0.45, to=0.55, by=0.05)
sens.results <- NULL
data("song.data")
data("glo.parms")
years=2005-1969
iteration=5
timestep=1
n.territories <- glo.parms$n.territories
starting.trait <- subset(song.data, Population=="Bear Valley" & Year==1969)$Trill.FBW
starting.trait2 <- c(starting.trait, rnorm(n.territories-length(starting.trait),
                                    mean=mean(starting.trait), sd=sd(starting.trait)))
init.inds <- data.frame(id = seq(1:n.territories), age = 2, trait = starting.trait2)
init.inds$x1 <-  round(runif(n.territories, min=-122.481858, max=-122.447270), digits=8)
init.inds$y1 <-  round(runif(n.territories, min=37.787768, max=37.805645), digits=8)

# Now we call the par.sens function with our specifications.
extra_parms <- list(init.inds = init.inds,
                    timestep = 1,
```

```
                    n.territories = nrow(init.inds),
                    learning.method = "integrate",
                    integrate.dist = 0.1,
                    lifespan = NA,
                    terr.turnover = 0.5,
                    mate.comp = FALSE,
                    prin = FALSE,
                    all = TRUE)
global_parms_key <- which(!names(glo.parms) %in% names(extra_parms))
extra_parms[names(glo.parms[global_parms_key])]=glo.parms[global_parms_key]
par.sens1 <- par.sens(parm = parm, par.range = par.range,
                      iteration = iteration, steps = years, mate.comp = FALSE,
                      fixed_parms=extra_parms[names(extra_parms)!=parm], all = TRUE)



#### Examine par.sens results
# Examine results objects, which include two arrays:

# The first array, `sens.results`, contains the SongEvo model results for each
# parameter. It has the following dimensions:
dimnames(par.sens1$sens.results)

# The second array, `sens.results.diff` contains the quantile range of trait
# values across iterations within a parameter value. It has the following
# dimensions:
dimnames(par.sens1$sens.results.diff)

# To assess sensitivity of SongEvo to a range of parameter values, plot the
# range in trait quantiles per year by the parameter value. We see that
# territory turnover values of 0.4--0.6 provided means and quantile ranges of
# trill bandwidths that are similar to those obtained with the published
# estimate of 0.5, indicating that the Acquire a Territory submodel is robust to
# realistic variation in those parameter values.

# In the figure, solid gray and black lines show the quantile range of song
# frequency per year over all iterations as parameterized with the published
# territory turnover rate (0.5; thick black line) and a range of values from 0.4
# to 0.6 (in steps of 0.05, light to dark gray). Orange lines show the mean and
# 2.5th and 97.5th quantiles of all quantile ranges.

# plot of range in trait quantiles by year by parameter value
plot(1:years, par.sens1$sens.results.diff[1,],
     ylim=range(par.sens1$sens.results.diff, na.rm=TRUE),
     type="l",
     ylab="Quantile range (Hz)", xlab="Year",
     col="transparent", xaxt="n")
axis(side=1, at=seq(0, 35, by=5), labels=seq(1970, 2005, by=5))
  #Make a continuous color ramp from gray to black
grbkPal <- colorRampPalette(c('gray','black'))

  #Plot a line for each parameter value
for(i in 1:length(par.range)){
```

```
lines(1:years, par.sens1$sens.results.diff[i,],
      col=grbkPal(length(par.range))[i])
}
  #Plot values from published parameter values
lines(1:years, par.sens1$sens.results.diff[2,], col="black", lwd=4)
  #Calculate and plot mean and quantiles
quant.mean <- apply(par.sens1$sens.results.diff, 2, mean, na.rm=TRUE)
lines(quant.mean, col="orange")
#Plot 95% quantiles (which are similar to credible intervals)
  #95% quantiles of population means (narrower)
quant.means <- apply(par.sens1$sens.results.diff, MARGIN=2,
                     quantile, probs=c(0.975, 0.025), R=600, na.rm=TRUE)
lines(quant.means[1,], col="orange", lty=2)
lines(quant.means[2,], col="orange", lty=2)
```

`song.data` *White Crown Sparrow Song Observations*

Description

A dataset of mean trill bandwidth from 3 WCSP male populations in 1969 and 2005

Usage

```
song.data
```

Format

A data frame with 89 rows and 3 variables:

Population Locality of the observed male sparrow

Year Year in which samples were taken

Trill.FBW Mean observed trill bandwidth

`SongEvo` *Model bird song evolution*

Description

This function simulates bird song evolution. Submodels are performed once per time step, and include fledging from the nest, song learning, ageing and death, dispersal, competition for territories, mate attraction, and reproduction.

Usage

```
SongEvo(init.inds, iteration, steps, timestep, terr.turnover, mate.comp,
  learning.method, integrate.dist, learning.error.d, learning.error.sd,
  mortality.a, mortality.j, lifespan, phys.lim.min, phys.lim.max,
  male.fledge.n.mean, male.fledge.n.sd, male.fledge.n, disp.age,
  disp.distance.mean, disp.distance.sd, n.territories, prin, all)
```

Arguments

`init.inds`	Initial population data. A data frame that includes columns for "id," "age," "trait," "x1" (longitude) and "y1" (latitude).
`iteration`	The number of iterations that the model will run.
`steps`	The number of steps per iteration.
`timestep`	The length of time that passes in each step. For annually breeding species, timestep = 1 year.
`terr.turnover`	The proportion of territories that change ownership during a step.
`mate.comp`	Female preference for mates. Options are TRUE or FALSE.
`learning.method`	If an individual learns from their ("father") or all males within a specified radius ("integrate").
`integrate.dist`	Distance over which song learning is integrated.
`learning.error.d`	Direction of learning error.
`learning.error.sd`	The standard deviation of imitation error.
`mortality.a`	Annual mortality of adults (after the first time step).
`mortality.j`	Annual mortality of juvenile birds (in the first time step).
`lifespan`	Maximum age for individuals; any number is accepted. "NA" causes SongEvo to disregard lifespan and sets population size based on mortality rates alone.
`phys.lim.min`	The minimum physical limit of trait production.
`phys.lim.max`	The maximum physical limit of trait production.
`male.fledge.n.mean`	The mean number of offspring produced per time step per individual breeding male. Includes only offspring raised in that breeding male's nest (i.e. it does not account for extra-pair offspring in other nests).
`male.fledge.n.sd`	Standard deviation of the number of male fledglings.
`male.fledge.n`	A vector of the number of offspring for the initial population, optionally calculated with male.fledge.n.mean and male.fledge.n.sd
`disp.age`	The age at which individual males disperse from their birth location.
`disp.distance.mean`	The distance that individual males disperse (meters).

`disp.distance.sd`
: The standard deviation of dispersal distance.

`n.territories`
: The number of territories in the population. This number is fixed for all iterations.

`prin`
: Print summary values after each timestep has completed? Options are TRUE or FALSE.

`all`
: Save data for all individuals? Options are TRUE or FALSE.

Value

three objects. First, currently alive individuals are stored in a data frame called "inds." Values within "inds" are updated throughout each of the iterations of the model, and "inds" can be viewed after the model is completed. Second, an array (i.e. a multi-dimensional table) entitled "summary.results" includes population summary values for each time step (dimension 1) in each iteration (dimension 2) of the model. Population summary values are contained in five additional dimensions: population size for each time step of each iteration ("sample.n"), the population mean and variance of the song feature studied ("trait.pop.mean" and "trait.pop.variance"), with associated lower ("lci") and upper ("uci") confidence intervals. Third, individual values may optionally be concatenated and saved to one data frame entitled "all.inds." all.inds can become quite large, and is therefore only recommended if additional data analyses are desired.

See Also

`par.sens`, `par.opt`, `mod.val`, `h.test`

Examples

```
### See vignette for an example that uses all functions in SongEvo.

### Load the example data: song.data

# To explore the SongEvo package, we will use a database of songs from Nuttall's
# white-crowned sparrow (*Zonotrichia leucophrys nuttalli*) recorded at three
# locations in 1969 and 2005.

data("song.data")

# Examine global parameters.Global parameters describe our understanding of the
# system and may be measured or hypothesized. They are called "global" because
# they are used by many many functions and subroutines within functions. For
# descriptions of all adjustable parameters, see `?WCSP` or Danner et al. (year)
str(glo.parms)

# Share global parameters with the global environment. We make these parameters
# available in the global environment so that we can access them with minimal
# code.
list2env(glo.parms, globalenv())

#### Examine song data
```

```
# Data include the population name (Bear Valley, PRBO, or Schooner), year of
# song recording (1969 or 2005), and the frequency bandwidth of the trill.
str(song.data)

### Simulate bird song evolution with `SongEvo()`

#### Define initial individuals

# In this example, we use songs from individual birds recorded in one population
# (PRBO) in the year 1969, which we will call `starting.trait`.
starting.trait <- subset(song.data, Population=="PRBO" & Year==1969)$Trill.FBW

# We want a starting population of 40 individuals, so we generate additional
# trait values to complement those from the existing 30 individuals. Then we
# create a data frame that includes a row for each individual; we add
# identification numbers, ages, and geographical coordinates for each
# individual.
starting.trait2 <- c(starting.trait, rnorm(n.territories-length(starting.trait),
                                           mean=mean(starting.trait),
                                           sd=sd(starting.trait)))
init.inds    <- data.frame(id = seq(1:n.territories), age = 2, trait = starting.trait2)
init.inds$x1 <- round(runif(n.territories, min=-122.481858, max=-122.447270), digits=8)
init.inds$y1 <- round(runif(n.territories, min=37.787768, max=37.805645), digits=8)

#### Specify and call the SongEvo model

# SongEvo() includes several settings, which we specify before running the
# model. For this example, we run the model for 10 iterations, over 36 years
# (i.e. 1969--2005). When conducting research with `SongEvo()`, users will want
# to increase the number iterations (e.g. to 100 or 1000). Each timestep is one
# year in this model (i.e. individuals complete all components of the model in 1
# year). We specify territory turnover rate here as an example of how to adjust
# parameter values.  We could adjust any other parameter value here also. The
# learning method specifies that individuals integrate songs heard from adults
# within the specified integration distance (intigrate.dist, in kilometers). In
# this example, we do not includ a lifespan, so we assign it NA. In this
# example, we do not model competition for mates, so specify it as FALSE. Last,
# specify all as TRUE in order to save data for every single simulated
# individual because we will use those data later for mapping. If we do not need
# data for each individual, we set all to FALSE because the all.inds data.frame
# becomes very large!
iteration <- 5
years <- 36
timestep <- 1
terr.turnover <- 0.5
learning.method <- "integrate"
integrate.dist <- 0.1
lifespan <- NA
mate.comp <- FALSE
prin <- FALSE
all <- TRUE

# Now we call SongEvo with our specifications and save it in an object called
```

```
# SongEvo1.
SongEvo1 <- SongEvo(init.inds = init.inds,
                    iteration = iteration,
                    steps = years,
                    timestep = timestep,
                    n.territories = n.territories,
                    terr.turnover = terr.turnover,
                    learning.method = learning.method,
                    integrate.dist = integrate.dist,
                    learning.error.d = learning.error.d,
                    learning.error.sd = learning.error.sd,
                    mortality.a = mortality.a,
                    mortality.j = mortality.j,
                    lifespan = lifespan,
                    phys.lim.min = phys.lim.min,
                    phys.lim.max = phys.lim.max,
                    male.fledge.n.mean = male.fledge.n.mean,
                    male.fledge.n.sd = male.fledge.n.sd,
                    male.fledge.n = male.fledge.n,
                    disp.age = disp.age,
                    disp.distance.mean = disp.distance.mean,
                    disp.distance.sd = disp.distance.sd,
                    mate.comp = mate.comp,
                    prin = prin,
                    all)

#### Examine results from SongEvo model

# The model required the following time to run on your computer:
SongEvo1$time


# Three main objects hold data regarding the SongEvo model.  Additional objects
# are used temporarily within modules of the model.

# First, currently alive individuals are stored in a data frame called “inds.”
# Values within “inds” are updated throughout each of the iterations of the
# model, and “inds” can be viewed after the model is completed.
head(SongEvo1$inds, 5)

# Second, an array (i.e. a multi-dimensional table) entitled “summary.results”
# includes population summary values for each time step (dimension 1) in each
# iteration (dimension 2) of the model.  Population summary values are contained
# in five additional dimensions: population size for each time step of each
# iteration (“sample.n”), the population mean and variance of the song feature
# studied (“trait.pop.mean” and “trait.pop.variance”), with associated lower
# (“lci”) and upper (“uci”) confidence intervals.
dimnames(SongEvo1$summary.results)

# Third, individual values may optionally be concatenated and saved to one data
# frame entitled “all.inds.”  all.inds can become quite large, and is therefore
# only recommended if additional data analyses are desired.
head(SongEvo1$all.inds, 5)
```

```
#### Simulated population size

# We see that the simulated population size remains relatively stable over the
# course of 36 years. This code uses the summary.results array.
plot(SongEvo1$summary.results[1, , "sample.n"],
     xlab="Year", ylab="Abundance",
     type="n", xaxt="n",
     ylim=c(0, max(SongEvo1$summary.results[, , "sample.n"], na.rm=TRUE)))
axis(side=1, at=seq(0, 40, by=5), labels=seq(1970, 2010, by=5))
for(p in 1:iteration){
lines(SongEvo1$summary.results[p, , "sample.n"], col="light gray")
}
n.mean <- apply(SongEvo1$summary.results[, , "sample.n"], 2, mean, na.rm=TRUE)
lines(n.mean, col="red")
#Plot 95% quantiles
quant.means <- apply(SongEvo1$summary.results[, , "sample.n"],
                     MARGIN=2,
                     quantile, probs=c(0.975, 0.025), R=600, na.rm=TRUE)
lines(quant.means[1,], col="red", lty=2)
lines(quant.means[2,], col="red", lty=2)

# Load Hmisc package for plotting functions.
library("Hmisc")

#### Simulated trait values

# We see that the mean trait values per iteration varied widely, though mean
# trait values over all iterations remained relatively stable. This code uses
# the summary.results array.
plot(SongEvo1$summary.results[1, , "trait.pop.mean"],
     xlab="Year", ylab="Bandwidth (Hz)",
     xaxt="n", type="n",
    xlim=c(-0.5, 36), ylim=range(SongEvo1$summary.results[, , "trait.pop.mean"], na.rm=TRUE))
for(p in 1:iteration){
lines(SongEvo1$summary.results[p, , "trait.pop.mean"], col="light gray")
}
freq.mean <- apply(SongEvo1$summary.results[, , "trait.pop.mean"], 2, mean, na.rm=TRUE)
lines(freq.mean, col="blue")
axis(side=1, at=seq(0, 35, by=5), labels=seq(1970, 2005, by=5))#, tcl=-0.25, mgp=c(2,0.5,0))
#Plot 95% quantiles
quant.means <- apply(SongEvo1$summary.results[, , "trait.pop.mean"],
                     MARGIN=2,
                     quantile, probs=c(0.95, 0.05), R=600, na.rm=TRUE)
lines(quant.means[1,], col="blue", lty=2)
lines(quant.means[2,], col="blue", lty=2)
#plot mean and CI for historic songs.
 #plot original song values
library("boot")
sample.mean <- function(d, x) {
mean(d[x])
}
boot_hist <- boot(starting.trait, statistic=sample.mean, R=100)#, strata=mn.res$iteration)
```

```
ci.hist <- boot.ci(boot_hist, conf=0.95, type="basic")
low <- ci.hist$basic[4]
high <- ci.hist$basic[5]
points(0, mean(starting.trait), pch=20, cex=0.6, col="black")
errbar(x=0, y=mean(starting.trait), high, low, add=TRUE)
 #text and arrows
text(x=5, y=2720, labels="Historical songs", pos=1)
arrows(x0=5, y0=2750, x1=0.4, y1=mean(starting.trait), length=0.1)

#### Trait variance

# We see that variance for each iteration per year increased in the first few
# years and then stabilized. This code uses the summary.results array.

#plot variance for each iteration per year
plot(SongEvo1$summary.results[1, , "trait.pop.variance"],
     xlab="Year", ylab="Bandwidth Variance (Hz)",
     type="n", xaxt="n",
     ylim=range(SongEvo1$summary.results[, , "trait.pop.variance"], na.rm=TRUE))
axis(side=1, at=seq(0, 40, by=5), labels=seq(1970, 2010, by=5))
for(p in 1:iteration){
lines(SongEvo1$summary.results[p, , "trait.pop.variance"], col="light gray")
}
n.mean <- apply(SongEvo1$summary.results[, , "trait.pop.variance"], 2, mean, na.rm=TRUE)
lines(n.mean, col="green")
#Plot 95% quantiles
quant.means <- apply(SongEvo1$summary.results[, , "trait.pop.variance"],
                     MARGIN=2,
                     quantile, probs=c(0.975, 0.025), R=600, na.rm=TRUE)
lines(quant.means[1,], col="green", lty=2)
lines(quant.means[2,], col="green", lty=2)

#### Maps

# The simulation results include geographical coordinates and are in a standard
# spatial data format, thus allowing calculation of a wide variety of spatial
# statistics.

# Load packages for making maps.
library("reshape2")
library("lattice")
library("sp")

# Convert data frame from long to wide format. This is necessary for making a
# multi-panel plot.
all.inds1 <- subset(SongEvo1$all.inds, iteration==1)
w <- dcast(as.data.frame(all.inds1), id ~ timestep, value.var="trait", fill=0)
all.inds1w <- merge(all.inds1, w, by="id")
names(all.inds1w) <- c(names(all.inds1), paste("Ts", seq(1:years), sep=""))

# Create a function to generate a continuous color palette--we will use the
# palette in the next call to make color ramp to represent the trait value.
```

```
rbPal <- colorRampPalette(c('blue','red'))

# Plot maps, including a separate panel for each timestep (each of 36 years).
# Our example shows that individuals move across the landscape and that regional
# dialects evolve and move. The x-axis is longitude, the y-axis is latitude, and
# the color ramp indicates trill bandwidth in Hz.
spplot(all.inds1w[,-c(1:ncol(all.inds1))],
       as.table=TRUE,
       cuts=c(0, seq(from=1500, to=4500, by=10)),
       ylab="",
    # cuts specifies that the first level (e.g. <1500) is transparent.
       col.regions=c("transparent", rbPal(1000)),
       colorkey=list(
         right=list(
           fun=draw.colorkey,
           args=list(
             key=list(
               at=seq(1500, 4500, 10),
               col=rbPal(1000),
               labels=list(at=c(1500, 2000, 2500, 3000, 3500, 4000, 4500),
                        labels=c("1500", "2000", "2500", "3000", "3500", "4000", "4500")
               )
             )
           )
         )
       )
)

# In addition, you can plot simpler multi-panel maps that do not take advantage
# of the spatial data class.

# Lattice plot (not as a spatial frame)
it1 <- subset(SongEvo1$all.inds, iteration==1)
# Create a function to generate a continuous color palette
rbPal <- colorRampPalette(c('blue','red'))
it1$Col <- rbPal(10)[as.numeric(cut(it1$trait, breaks = 10))]
xyplot(it1$y1~it1$x1 | it1$timestep,
       groups=it1$trait, asp="iso", col=it1$Col,
       xlab="Longitude", ylab="Latitude")
```

Index