Package 'VerDi'

October 17, 2018

Title Modelling the vertical distribution of aquatic organisms

Version 0.1

Description

Predicting the vertical distribution of pelagic and semi-pelagic fish and zooplankton. Model approach is based on Evaluation Functions (rule-based systems). Required knowledge concerns the species' habitat preferences represented as environmental factors, i.e. abiotic or biotic parameter thresholds (e.g. water temperature, irradiation, food availability, etc.). Further functions (e.g. modelling underwater light regimes based on remote sensing) are included...

Depends R (>= 3.5.1)

License What license is it under?

Encoding UTF-8

LazyData true

Maintainer Robert Herrmann <robertherrmann@mail.de>

Author Robert Herrmann

RoxygenNote 6.1.0

Import data.table

NeedsCompilation no

R topics documented:

`EvalFunc` *Evaluation function*

Description

Function computes the organism's vertical movement based on a set of single environmental factors.

Usage

```
EvalFunc(hydro, faclist, nInd, mSpeed = 1, tStep = 1, rWalk = c("ND",
  0, 0.1, 0.1), dimList)
```

Arguments

`hydro`	data frame including set of environmental parameters.
`faclist`	data frame specifying the species specific thresholds that induce a vertical movement.
`nInd`	numeric, number of individuals that are included in modelled environment.
`mSpeed`	numeric, maximum vertical distance that can be reached per time interval, Default: 1.
`tStep`	numeric, number of time increments, Default: 1.
`dimList`	numeric, intended return. See 'Details'.
`rwalk`	numeric vector indicating the demographic noise of the modelled population. A pre-defined random walk is included by default. See 'Details'.
`rWalkOff`	logical, if TRUE, demographic noise is turned off. If FLASE, a pre-defined random walk is included (default). See 'Details'.

Details

The arguments hydro and faclist should be of class data.frame. Note that colnames of hydro and faclist have to be equal. Required layouts are given in 'Examples'.

If the argument nInd is provided, the model output includes the position of each individual for each time increament. The individuals are randomly distributed over the entire water column when the calculations begin. Note that computation time is strongly correlated with the amount of individuals included in the model environment.

Note that the arguments mSpeed and tStep are linked, i.e. mSpeed is the vertical distance each individual can reach in a time interval given by tStep. For example, if you aim to predict the vertical ditribution for an entire day with one-minute-intervals, than tStep is set to 1440 minutes, and consequently, mSpeed must be converted to meter per minute. For further details see 'Examples'.

The argument dimList provides three diffrent options of what specific data type is returned. Further details need to be written. See 'Examples'.

Function makes use of RWalk. Default: c("ND", 0, 0.1, 0.1). If rWalk is set to NULL, the random walk is turned off and each individual would remain at the given water depth as long as the ambient environment stays sufficient enough.

Further details in 'Examples'.

Examples

```
## Not run:

#-1-----Example for static hydrography (stat.hyd) and given factor list (fl)
#-1.1---Generating hydrography commonly found in the Bornholm Basin (Baltic Sea)
set.seed(123456)
stat.hyd <- HydroGen(data.frame(Temp = c(17,13,4,4.8,8),
                                Ox = c(6,5,3,2,0),
                                Sal = c(7,8,9,18,20)), by = 25)

#-1.2---Generating factor list for any kind of organism
fac  <- c("UL", "LL", "MOV", "RI")
temp <- c(  10,    8,    -1,  0.33)
ox   <- c(   1,  0.5,     1,  0.33)
sal  <- c(  10,   11,    -1,  0.33)
fl   <- data.frame(rbind(temp, ox, sal))
colnames(fl) <- fac
rownames(fl) <- colnames(stat.hyd)

#-1.3.1-Executing EvalFunc(defaults) and calculating free vertival range of organisms
output.a <- EvalFunc(hydro = stat.hyd, faclist = fl)
FVR.a    <- which(output.a["PredMovByAllPar",] == 0)

#-1.3.2-Alternatively use EvalFunc(dimList = 2) for extracting "PredMovByAllPar"
output.b <- EvalFunc(hydro = stat.hyd, faclist = fl, dimList = 2)
FVR.b    <- which(output.b$PredMovByAllPar == 0)

#-1.4---Plotting
par(mfrow = c(1,1))
df <- as.data.frame(output.a)
plot(x = 0, ylim = c(nrow(df), 0), xlim = c(min(df[,1:3]), max(df[,1:3])),
   type = "n", ylab = "Water depth [m]", main = "Hydrography and free vertical range (shaded area)",
     xlab = "Temperature (T) / Oxygen (O) / Salinty (S)")
rect(0, max(FVR.a), 20, min(FVR.a), angle = 45, density = 6, col = "grey70")
for(i in 1:3){
  c <- c("red", "blue", "darkgreen")
  lines(df[,i], as.numeric(rownames(df)), lty = i, lwd = 2, col = c[i])
  posY <- round(nrow(df) * 0.9, 0)
  points(df[posY,i], posY, pch = 21, bg = "white", cex = 4)
  text(df[posY,i], posY, substr(colnames(df)[i], 1, 1))
}

#-2-----Example for dynamic hydrography (DynHyd) including intra daily light regimes
#-2.1---Generating interpolated hydrography profiles for CTDs by use of InterPro()
set.seed(123)
hyd.a <- HydroGen(data.frame(Temp = c(12,7,5,7,8,9), Ox = c(8,7,5,3,2,0)), by = 20)
hyd.b <- HydroGen(data.frame(Temp = c(10,7,4.8,7,8), Ox = c(8,7,6,5,0)), by = 25)
hyd.c <- HydroGen(data.frame(Temp = c(11,4,8), Ox = c(8,4,0)), by = 50)
hyd.d <- HydroGen(data.frame(Temp = c(11,7,4,6,8), Ox = c(8,7,6,4,0)), by = 25)
hyds <- list(hyd.a, hyd.b, hyd.c, hyd.d)
tp <- matrix(c(1,2,2,3,3,4), ncol = 3)
dyn.hyd <- list()
for(i in 1:3){dyn.hyd <- c(dyn.hyd, InterPro(data = list(hyds[[tp[1,i]]], hyds[[tp[2,i]]]),
                                             steps = 59, opList = TRUE))}

#-2.2---Generating light regimes (19:00 - 22:00) by use of date-at-location specific regression
```

```
light <- LightBornholm(depth   = c(0:99), ATTk = -0.16, lx = TRUE,
                       daytime = seq(19,22,0.01666667), min = 2)

for(i in 1:180){dyn.hyd[[i]] <- cbind(dyn.hyd[[i]], light[,i], light[,i])
colnames(dyn.hyd[[i]]) <- c(colnames(hyd.a), "UpperLight", "LowerLight")}

hyd <- array(unlist(dyn.hyd), dim = c(dim(dyn.hyd[[1]]), length(dyn.hyd)),
             dimnames = list(NULL,colnames(dyn.hyd[[1]]), NULL))

#-2.3---Generating factor list for any kind of organism (e.g. European sprat)
fac      <- c("UL",  "LL", "MOV", "RI")
temp     <- c(  4,     5,      0,  0.5)
ox       <- c(   1,   0.5,     1,  0.5)
lightUp  <- c( 500,    10,    -1,  0.2)
lightLo  <- c( 0.1, 0.005,     1,  0.2)
fl       <- data.frame(rbind(temp, ox, lightUp, lightLo))
colnames(fl) <- fac
rownames(fl) <-  colnames(hyd)

#-2.3---Executing EvalFunc() for given hydro (including light regimes) and factor list
df     <- EvalFunc(hydro = hyd, faclist = fl)
head(df[,,1],10)

#-2.4---Plotting
lists  <- EvalFunc(hydro = hyd, faclist = fl, dimList = 2)
PVD    <- lists$PredVerDi
image  <- as.raster(PVD)
time   <- format(seq(from = as.POSIXct("2001-06-04 19:00"),
                     to = as.POSIXct("2001-06-04 22:00"),
                     by = "min"), "%H:%M")[c(1,31,61, 91,121,151,181)]

AS <- function(m) t(m)[,nrow(m):1]
FS <- function(x) (x-min(x))/(max(x)-min(x))

layout(matrix(c(1,2,3), 3, 1),
       widths=c(1,1,1), heights=c(1,1,2))

#-2.4.1-Temperature profile
par(mar = c(0.5,4.1,2.1,2.1))
colfunc <- colorRampPalette(c("blue1", "lightblue", "limegreen", "yellowgreen",
                              "yellow", "orange", "red"))
image(AS(lists$Temp), useRaster=TRUE, col = colfunc(500), axes = FALSE, ylab = "Water depth [m]")
legend(-0.04, 1.05, "TEMP",  bty="n", cex = 1.8, xjust = 0, yjust = 1)
abline(v = seq(0,1,length.out = 7), h = seq(0,1,0.2), col = "grey70", lty = 3)
axis(2, at = seq(0,1,0.2), labels = seq(100,0,-20), las = 2); box()

#-2.4.2-Oxygen profile
par(mar = c(2,4.1,0.5,2.1))
colfunc <- colorRampPalette(c("slategray4", "slategray", "slategray3", "slategray2",
                              "slategray1", "white", "orangered1"))
image(AS(lists$Ox), useRaster=TRUE, col = rev(colfunc(500)), axes = FALSE, ylab = "Water depth [m]")
legend(-0.04, 1.05, "OX",  bty="n", cex = 1.8, xjust = 0, yjust = 1)
abline(v = seq(0,1,length.out = 7), h = seq(0,1,0.2), col = "grey70", lty = 3)
axis(2, at = seq(0,1,0.2), labels = seq(100,0,-20), las = 2); box()

#-2.4.2-Predicted vertical distribution of sprat - high probabilty (bright araes) vs. low prob (dark areas)
par(mar = c(5.1,4.1,0.5,2.1))
```

```
colfunc <- colorRampPalette(c("black", "white"))
image(AS(lists$PredVerDi), useRaster=TRUE, col = colfunc(500), axes = FALSE, ylab = "Water depth [m]",
      xlab = "Day time [UTC+2]")
legend(-0.04, 1.03, "MODEL",  bty="n", cex = 1.8, xjust = 0, yjust = 1)
abline(v = seq(0,1,length.out = 7), h = seq(0,1,0.2), col = "grey70", lty = 3)
axis(2, at = seq(0,1,0.2), labels = seq(100,0,-20), las = 2)
axis(1, at = seq(0,1,length.out = 7), labels = time); box()

#-2.4.2-Adding movement pattern for 60 individuals of European sprat
set.seed(12345)
nIndivi <- 60
testInd <- EvalFunc(hydro = hyd, faclist = fl, nInd = nIndivi, mSpeed = 3, rWalk = c("ND", 0, 0.1, 0.03))
colfunc <- colorRampPalette(c("blue", "limegreen", "yellow", "orange", "red"))
for(i in 1:nIndivi){
  lines(FS(c(1:length(testInd[i,]))), 1-testInd[i,]/nrow(lists$PredVerDi),
        col = colfunc(nIndivi)[i], cex = 1.2)
}

par(mfrow = c(1,1))

## End(Not run)
```

`GeneralArealOverlap` *General areal overlap*

Description

Function calculates interspecific overlap among two populations.

Usage

```
GeneralArealOverlap(data)
```

Arguments

`data`	matrix or data frame including two colums - one for each population's distribution.

Details

The general areal overlap is defined as the area occupied by both populations devided by the area occupied by either one or the other population.

Examples

```
#Example shows how two populations overlap in one of five water layers

df <- cbind(c(1,1,1,0,0),
            c(0,0,1,1,1))

GeneralArealOverlap(df)
```

HydroGen *Hydrography generator*

Description

Function generates customized hydrography for test purposes.

Usage

```
HydroGen(data, by = 10)
```

Arguments

data	data frame including environmental parameters.
by	numeric, interval between each data point, Default: 10.

Details

Generates a customized hydrography. More details in 'Examples'.

Examples

```
#Example generates hydro including temperature, salinity and oxygen

data <- data.frame(Temperature = c(17, 15, 10,   8,  4, 4.5,   6,  8),
                   Oxygen =      c(6,   5,  4,   3,  2,   1, 1.5,  0),
                   Salinity =    c(7, 7.5,  8, 8.5,  9, 9.5,  14, 20))

HydroGen(data = data, by = 10)
```

InterPro *Hydrography interpolator*

Description

Function computes linear interpolation between two hydrographies.

Usage

```
InterPro(data, steps, opList = FALSE)
```

Arguments

data	list of two hydrography profiles.
steps	numeric, number of profiles to be interpolated.
opList	logical, if TRUE, returns list. If FALSE, returns array.

Details

To be written. Further details in 'Examples'.

Examples

```
## Not run:

#Example for two customized CTDs along south-to-north transect (20 nautical miles)
#Computation of linearly interpolated hydrographies for every nautical mile

#Generating two customized hydrographies
set.seed(12345)
hydNorth <- HydroGen(data.frame(Temperature = c(17, 10, 4, 7, 9),
                                Oxygen = c(6, 5, 3, 2, 0),
                                Salinity = c(7, 8, 9, 18, 20)), by = 20)

hydSouth <- HydroGen(data.frame(Temperature = c(20, 17, 3, 7, 8),
                                 Oxygen = c(7, 6, 4, 1, 0),
                                 Salinity = c(6, 8, 9, 16, 20)), by = 20)

#Generating list including hydNorth and hydSouth
dl  <- list(hydNorth, hydSouth)

#Calculating nineteen interpolated profiles
hydro <- InterPro(data = dl, steps = 19, opList = TRUE)

#Generating plot including temperature and salinity profiles from southern towards northern station
par(mfrow = c(1,2))
for(j in c(1,3)){
 plot(x = 0, xlim = c(-20, 20), ylim = c(85,0), ylab = ifelse(j == 1, "Water depth", ""), xlab = "", xaxt='n', typ
  points(c(-18, 18), c(84, 84), pch = 21, cex = 4)
  text(c(-18, 18), c(84, 84), c("S", "N"), cex = 1.5)
  title(main = c("Interpolated temperature profiles\nfrom one station to another", "",
                 "Interpolated salinity profiles\nfrom one station to another")[j])

  for(i in 20:1){
    lines(hydro[[i]][,j]-i, c(1:80), col = c("red3", "", "green4")[j])
  }
 }

## End(Not run)
```

`LightBornholm` *Light regime generator*

Description

Function generates semi-customized light profile typically found in the Bornholm Basin (Central Baltic Sea) in early May.

Usage

```
LightBornholm(depth, daytime = seq(0, 24, 0.01666667), ATTk, min, lx)
```

Arguments

`depth`	numeric vector, depth profile which is used by light model, Default: c(1:100).

daytime	numeric vector, time sequence in decimals, e.g. every single minute over 24 hours (default).
ATTk	numeric, attenuation coefficient.
min	numeric, minimum illumination at shallowest depth.
lx	logical, if TRUE, output is in lux [lx], if FALSE, computations are in W/m2 (default).

Details

Function makes use of light model developed by (...). Further details in 'Examples'.

Examples

```
## Not run:

#The minimum illumination at >0m of depth is assumed to be 2 lx
minVal <- 2

#Attenuation coefficiant is set to -0.16, which can be considered as relatively 'clear'
atteCo <- -0.16

#Computing intra daily under water light regime [lx] down to 100m of depth in Bornholm Basin
testmatrix <- LightBornholm(depth = c(1:100), ATTk = atteCo, lx = TRUE, min = minVal)

#Assigning time of day to each column
ts <- format(seq(from = as.POSIXct("2017-08-19 0:00"), length.out = 1440, by = "min"), "%H:%M")
colnames(testmatrix) <- ts

#Printing underwater light regime between 20 and 40 meters of depth at 00:00, 06:00, 12:00 and 18:00
testmatrix[20:40, c("00:00", "06:00", "12:00", "18:00")]

## End(Not run)
```

---

parGrad *Calculating gradients*

Description

Function calculates in which direction and at what rate a certain parameter changes over depth.

Usage

```
parGrad(depth, par, range = 5)
```

Arguments

depth	numeric vector of depth indication.
par	numeric vector of single environmental parameter (e.g. water temperature).
range	numeric integer indicating the range for which the delta should be calculated, Default: 5.

Examples

```
#Generating customized temperature and salinity profile
hydro <- HydroGen(data.frame(temp = c(17,10,6,8,9),
                             salt = c(7,8,9,18,20)),
                             by = 25)

#Calculating salinity and temperature gradient on a 3-meter-range
hydro$tempGrad <- parGrad(depth = seq_along(hydro$temp), par = hydro$temp, range = 3)
hydro$saltGrad <- parGrad(depth = seq_along(hydro$salt), par = hydro$salt, range = 3)

head(hydro, 15)
```

`RWalk` *Demographic noise*

Description

Function generates random walk (demographic noise) based on a normal distribution (ND) or uniform distribution (UD).

Usage

```
RWalk(par, hs = 0.1, mv = 0, sd = 0.1, min = -0.2, max = 0.2,
  op = "ND")
```

Arguments

par	numeric vector or R object.
hs	numeric, half saturation parameter, Default: 0.1
mv	numeric, mean when op = "ND", Default: 0
sd	numeric, standard deviation when op = "ND", Default: 0.1
min	numeric, minimum value when op = "UD", Default: -0.2
max	numeric, maximum value when op = "UD", Default: 0.2
op	character, either 'ND' (normal distribution) or 'UD' (uniform distribution), Default: 'ND'

Details

Is used to add demograhic noise (random walk) to e.g. the population's vertical distribution. Further details in rnorm() and runif().

Examples

```
## Not run:

#Example shows difference between options (op)
#Variables (hs, mv, sd, min, max) are set to default (see description above)

rw <- data.frame()

set.seed(123)
```

```
for(i in 1:100){
 rw[i,1] <- RWalk(0)
 rw[i,2] <- RWalk(0, op = "UD")

 if(i == 100){colnames(rw) <- c("Normally distributed random walk", "Uniformly distributed random walk")

  par(mfrow = c(2, 2))
 plot(rw[,1], type = "l", main = colnames(rw)[1], ylab = "Random walk", xlab = "", ylim = c(-0.4, 0.4))
 plot(rw[,2], type = "l", main = colnames(rw)[2], ylab = "", xlab = "", ylim = c(-0.4, 0.4))
  hist(rw[,1], breaks = 100, main = "", xlab = "", xlim = c(-0.4, 0.4), ylim = c(0, 14))
  lines(density(rw[,1], adjust=2), lwd = 2)
 hist(rw[,2], breaks = 100, main = "", ylab = "", xlab = "", xlim = c(-0.4, 0.4), ylim = c(0, 14))
  lines(density(rw[,2], adjust=2), lwd = 2)
  }

 }

## End(Not run)
```

`SpecificArealOverlap` *Specific areal overlap*

Description

Function calculates fraction of areas occupied by one population on areas occupied by another population.

Usage

```
SpecificArealOverlap(data, col = 1)
```

Arguments

`data`	matrix or data frame including two colums - one for each population's distribution.
`col`	numeric integer indicating the column in which the species if interest in given, Default = 1.

Details

The specific areal overlap is defined as the area occupied by both populations devided by the area occupied by one of these populations.

Examples

```
#Example shows water column occupied by clupeid fish and cod

df <- data.frame(Clupeids = c(0,1,1,1,1,0),
                      Cod = c(0,0,0,1,1,1))

#Calculating specific areal overlap for clupeids and cod, respectively

clu <- SpecificArealOverlap(df, col = 1) * 100
```

```
cod <- SpecificArealOverlap(df, col = 2) * 100

paste(clu, "% of the water layers occupied by clupeid fish are also occupied by cod.", sep = "")
paste(cod, "% of the water layers occupied by cod are also occupied by clupeid fish.", sep = "")
```

TransFunc *Transition function*

Description

Function computes organism's vertical movement based on one single environmental factor.

Usage

```
TransFunc(par, UL, LL, MOV = -1, RI = 1)
```

Arguments

par	numeric vector of single environmental parameter.
UL	numeric, species specific parameter threshold located in upper water column.
LL	numeric, species specific parameter threshold located in lower water column.
MOV	numeric, direction of movement when threshold is reached, Default: -1.
RI	numeric, parameter weight, Default: 1.

Details

Code generates transition function and applies that to the vertical profile of a single environmental factor (e.g. water temperature). Further details in example below...

Examples

```
## Not run:

#Example for induced species' movement by ambient salinity and oxygen content (par)
#Generating customized salinity and oxygen profile by executing HydroGen()
set.seed(1231)
hydFic <- HydroGen(data = data.frame(Salinity = c(7,8,9,18,20),
                                     Oxygen = c(6,5,4,3,0)), by = 25)

#Generating a bunch of randomly distributed individuals (n = 20)
ind <- data.frame(runif(20, 1, nrow(hydFic)))

#Executing transition functions by applying TransFunc()
#Salinity: Maximum strength of downward movement (MOV = -1) when par <= upper limit (UL)
#Oxygen: Maximum strength of upward movement (MOV = 1) when par <= lower limit (LL)

movSal <- TransFunc(par = hydFic$Salinity, UL = 8, LL = 11)
movOx  <- TransFunc(par = hydFic$Oxygen, UL = 2, LL = 1, MOV = 1)

#Assigning strength and direction of movement to each individual
ind$movSal <- movSal[ind[,1]]
ind$movOx  <- movOx[ind[,1]]
```

```
colnames(ind) <- c("CurrentDepthOfIndividual", "PredictedMovementBy:Sal", "PredictedMovementBy:Ox")
head(ind, 10)

#Setting graphical paramaters
sig        <- c(25, 21, 24)
pa         <- cbind(hydFic$Salinity, hydFic$Oxygen)
col        <- c("limegreen", "blue")
ind$simSal <- sig[sign(movSal[ind[,1]])+2]
ind$simOx  <- sig[sign(movOx[ind[,1]])+2]
lsal     <- c(tail(which(hydFic$Salinity <= 8), 1), tail(which(hydFic$Salinity <= 11), 1))
lox        <- c(head(which(hydFic$Oxygen <= 2), 1), head(which(hydFic$Oxygen <= 1), 1))
limline    <- cbind(lsal, lox)
tit        <- c("Salinity profile including", "Oxygen profile including",
                "predicted individual movement\nand its strength")
xl         <- c("Practical salinity [PSU]", "Oxygen content [ml/l]")
yl         <- c("Water depth [m]", "")

#Plotting
par(mfrow = c(1,2),
    mar = c(5.1, 4.1, 4.1, 2.1))
for(i in 1:2){
 plot(pa[,i], c(1:length(pa[,i])), ylim = c(length(pa[,i]),1), xlab = xl[i], ylab = yl[i],
      type = "l", col = col[i], lwd = 2, main = paste(tit[i], tit[3], sep = "\n"))
  abline(h = limline[,i], col = "gray60", lty = 2)
  text(c(19, 5.5)[i], c(limline[1,i]+2, limline[2,i]+2), c("UL", "LL"))
  points(seq(min(pa[,i]), max(pa[,i]), length.out = 20), ind[,1],
         pch = ind[,i+3], cex = (ind[,i+1] * -1.3) ^ 4 + 2, bg = "grey")
  }

## End(Not run)
```

Index