Gabi Marin

Acabei de entrar no mestrado no Laboratório de Diversidade e Conservação de Mamíferos! O título do meu trabalho é Determinantes do Tamanho da Área de Vida e da Territorialidade em Pequenos mamíferos e é uma continuação da minha IC!

Página para exercícios

exec

Trabalho Final

Plano A:

A partir de uma tabela de pontos de captura de indivíduos, com coordenadas geográficas associadas, pretendo criar uma função que selecione quais desses indivíduos podem ser utilizados para uma análise de área de vida baseado em pelo menos três critérios: (1) número mínimo de capturas; (2) número de sessões de captura em que foram coletados; (3) excluir aqueles cujo os pontos não formam uma área (o que pode acontecer no caso de uma grade de captura). Além disso, a função deve ser capaz de calcular o tamanho da área de vida a partir do método MCP (mínimo polígono convexo).

Entrada: O objeto de entrada deve ser um arquivo (.csv) com pelo menos as colunas: indivíduo, ponto de captura, coordenada X, coordenada Y e sessão da captura.

Saída: Os objetos resultantes da função serão (1) um data.frame com um subset da tabela inicial com somente aqueles indivíduos que poderão ser analisados e todos os pontos em que foram capturados e (2) um data.frame com as colunas: indivíduo (não sei ainda se com todos os indivíduos os somente aqueles que cumpriram os critérios), área calculada, e, se possível, uma coluna com a sessão inicial de captura e outra com a sessão final. Caso haja mais características associadas a cada indivíduo (ex. sexo, idade, massa corpórea), eu gostaria que essas características ainda estivessem presentes no data.frame com a área de vida calculada (me esforçarei para isso).

Ps: Eu já sei que essa função talvez não rode em windows pois provavelmente dependerá do pacote gpclib que não dispõe de versão para Windows (mas já sei como resolver).

Plano B:

Criar uma função de lembretes dentro do R. A idéia é que a pessoa possa cadastrar lembretes sobre sua vida pessoal (ex. médico ou aniversários) para determinada data e quando essa data chegue, o lembrete seja plotado na área de gráficos. A função deve aceitar que vários lembretes sejam cadastrados ao mesmo tempo, com datas diferentes (no caso de coisas no mesmo dia, colocar no mesmo lembrete). Por enquanto, pelo o que eu consigo pensar, a pessoa terá que rodar todo dia a função quando abra o R (não sei ainda automatizar essa etapa). Além disso, essa função só é útil para

aquelas pessoas que utilizam o R diariamente, mas ainda acho válida.

Entrada: Não será necessário nenhum objeto. Nos próprios argumentos da função a pessoa deve cadastrar a data (arg1a) e o lembrete (arg1b). Para isso, a função terá muitos argumentos para que seja possível incluir muitos lembretes. Depois de um certo tempo, ela terá que atualizar seus lembretes.

Saída: O resultado da função é um gráfico com o recado no meio (ao invés de dados) caso haja um lembrete cadastrado para aquele dia ou então uma resposta automática, no próprio console do R, dizendo que não há lembretes.

Plano C (se puder):

Preferi ter um plano C também: uma versão simplificada do plano A. A partir de uma tabela de pontos de captura de indivíduos, com coordenadas geográficas associadas, pretendo criar uma função que selecione quais desses indivíduos podem ser utilizados para uma análise de área de vida baseado em pelo menos três critérios: (1) número mínimo de capturas; (2) número de sessões de captura em que foram coletados; (3) excluir aqueles cujo os pontos não formam uma área (o que pode acontecer no caso de uma grade de captura).

Entrada: O objeto de entrada deve ser um arquivo (.csv) com pelo menos as colunas: indivíduo, ponto de captura, coordenada X, coordenada Y e sessão da captura.

Saída: O objeto resultante da função será um data.frame com um subset da tabela inicial com somente aqueles indivíduos que poderão ser analisados e todos os pontos em que foram capturados.

Ps: Essa função não depende do pacote descrito no plano A e, portanto, poderá ser executada em windows.

> Gabi, achei tanto a proposta A quanto a B interessantes. Para calcular as áreas dos políginos, além do pacote que você mencionou, existe o pacote “sp” que traz uma série de funções para se trabalhar com poligonos. Outra coisa, acho melhor sua função receber um data.frame do que um arquivo csv, dessa forma o usuário pode armazenar os dados da forma que ele achar melhor e ainda assim vai poder usar sua função.
>
> Sobre a proposta B, achei muito legal, apenas note que a sua função tem que guardar os compromissos em algum tipo de arquivo. Além disso, talvez fosse bom se ela apagasse compromissos antigos depois de um tempo. —- *Danilo G. Muniz*

Plano Final:

Plano A com modificações sugeridas e mais uma:

Na função final, eu acrescentei mais um elemento para o output: uma lista de data.frames (por percetual de pontos utilizados) com os pontos de cada id que formam o MPC. Acabei achando mais útil manter um data.frame do com o subconjunto de ids úteis e todas as informações associadas a esse id e outro data.frame com os valores de área de vida dos MCP. Assim, o output da função final é, no máximo uma lista com dois data.frames e mais uma lista.

Help da Função

```
selection.MCP                       packege: nenhum                    R
Documentation

Seleção de dados. Cálculo do índice de área de vida MPC
Description:
A função seleciona os IDs que poderão ser utilizados para o cálculo do
índice MPC
(mínimo polígono convexo)(em inglês, MCP: minimun convex polygon) de tamanho
de Área Vida
baseado em alguns fatores de interesse. Depois, ela calcula esse índice.
Essa função
depende do pacote gpclib, que não está disponível para o sistema operacional
Windows,
dependendo ou de Mac ou de Linux.

Usage:

selection.MCP (coords, id, session, minsession, mincapture, calculation =
TRUE,
                         percent=c(80,95,100), unin = c("m"), unout=c("m2"),
mcpshow=FALSE)

Arguments:

coords          Data.frame contendo pelo menos as coordenadas X e Y
(cartesianas) de cada ponto
            de captura. Caso haja mais informações no data.frame, as
coordenadas X e Y devem
            ocupar a primeira e segunda coluna do data.frame
respectivamente.

id              Vetor com valores ou caracteres que determine a identidade (
a qual indivíduo
                pertence) de cada ponto de captura ( a quem pertence aquele
ponto). Caso todos os
                pontos pertençam ao mesmo indivíduo, incluir um vetor com o
mesmo valor em todas
```

```
                as posições.

session         Vetor com as sessões em que foram realizadas cada captura.
Esse argumento é
                importante para garantir independencia temporal entre os
pontos de captura de
                cada indivíduo. Caso esse fator não seja relevante, incluir
um vetor de número
                1 com o comprimento igual ao número de linhas do data.frame
"coords"(nrow(coords)).


minssession     Valor numérico: número mínimo de sessões diferentes que os
ids devem ter sido
                capturados para poderem ter suas áreas de vida estimada pelo
método MCP. Não
                aceita conjunto numérico como o argumento ´percent´.

mincapture      Valor numérico: número mínimo de pontos de captura
diferentes que os ids devem
                ter sido capturados para poderem ter suas áreas de vida
estimada pelo método MCP.
                Não aceita conjunto numérico como o argumento ´percent´.

calculation     Lógico, indicando se a estimativa MPC da área de vida deve
ser realizado ou
                não. Se calculation=TRUE, realiza o cálculo.

percent         Valor ou conjunto numércio. É a informação de qual
porcentagem de pontos
                (sempre mantendo os mais próximos do centróide da área de
vida) que deve ser
                mantida para realizar a estimativa. Ou então, 100 menos a
proporção de
                outlier que deve ser excluida do cálculo do MCP.

unin            A unidade dos pontos de captura inseridos. Ou "m", para
metros (default), ou
                "km", para Kilometros.

unout           A unidade que as áreas devem ser calculadas. Ou "m2"
(default), para metros
                quadrados (default), "km2" para Kilometros ou "ha" para
hectares.

mcpshow         Lógico. Determina se será mostrada uma lista de data.frames
com os pontos de
                captura utilizados para "desenhar" o mínimo polígono
convexo. Se "=TRUE",
                exibe um terceiro elemento na lista final (outra lista). Se
```

```
"=FALSE", não mostra,
                embora ela seja produzida de qualquer maneira.


Details:

Mínimo polígono convexo é uma propriedade matemática de um conjunto de
pontos: é o polígono
que, unindo alguns dos pontos desse conjunto, contenha todos os outros
pontos dentro de sua
área e que, ao mesmo tempo, não possua nenhum ângulo côncavo (ângulo interno
maior que 180 graus).

O mínimo polígono convexo foi proposto em 1957, por Mohr, como uma
estimativa de área de vida
para animais e, em seu próprio trabalho, apontou a importância de fazer a
estimativa excluindo
os outliers (papel do argumento "percent" nessa função).

A função selection.MCP deve ser aplicada em um data.frame com pelo menos
duas colunas e três
linhas (1 ID com 3 capturas).

Dentro desta função foram utilizadas duas funções adaptadas do pacote
"adehabitat_1.8.12",
para que a presente função não dependesse também desse pacote (de difícil
instalação). São as
funções mcp() e mcp.area(). As adaptações foram no sentido de tornar essas
funções mais flexíveis
e adequadas à finalidade da função principal.

Novamente, essa função depende do pacote gpclib, que não está diponível para
o sistema
operacional Windows, dependendo ou de Mac ou de Linux (já foi testada em
ambos).

Values:

A função selection.MCP retorna, dependendo do argumento ´calculation´ e do
argumento
´mcpshow´, um data.frame(se calculation=FALSE), uma lista com dois
data.frames (se
calculation=TRUE e mcpshow=FALSE) e uma lista com os dois data.frames e uma
lista de
data.frames (se calculation=TRUE e mcpshow=TRUE). O primeiro elemento da
lista, que estará
presente mesmo se calculation=FALSE, será um subconjunto do data.frame
inicial (de acordo
com os critérios definidos nos argumentos ´session´, ´minsession´,
´mincapture´), com as
mesmas colunas que este, sendo no mínimo 2 colunas (coordenadas X e Y), mas
```

```
sem limite
máximo de colunas.

O segundo elemento da lista será outro data.frame com os valores das áreas de vida
estimadas pelo método MCP para cada nível de id (indivíduo) nas colunas, e para cada valor
de porcentagem de pontos (argumento ´percent´) que devem ser considerados no cálculo do
MPC nas linhas.

O terceiro elemento da lista será outra lista de data.frames com pontos de captura
de fato utilizados para estimativa de MPC (dependendo do argumento ´percent´). Para
cada valor de ´percent´ é adicionado um data.frame nessa segunda lista com os ´percent´
pontos mais próximos do centróide e que foram incluidos na análise. Se o cálculo não
foi realizado, esse resultado não aparecera mesmo que "mpcshow=TRUE".

É importante notar que mesmo após a seleção de dados realizada, ainda podem aparecer
áreas de vida com estimativa MCP=0. Isso acontece quando todas as coordenadas X ou
todas as Y têm um mesmo valor, formando portanto, uma linha. Ou ainda, quando as coordenadas
formam uma diagonal em relação ao plano cartesiano.

Author:

Gabriela de Lima Marin
gabi_lm88@hotmail.com

References:

Calenge, C. (2011). "Home Range Estimation in R: the adehabitatHR Package." Office national
de la classe et de la faune sauvage
Saint Benoist – 78610 Auffargis – France.

Weisstein, Eric W. "Convex Polygon." From MathWorld--A Wolfram Web Resource.
http://mathworld.wolfram.com/ConvexPolygon.html

Morh, C. O. Table of equivalent populations of North American small mammals.
American Midland Naturalist, v. 37, p. 223–249,  1947.


See Also:
```

```
packedge adehabitat, para análises de seleção de habitat
packadge sp, para análises de dados espaciais

Exemples:

#1
library(gpclib) #chamando o pacote
ako<-read.table("tab0akodoncamila.csv",sep=";",head=TRUE, as.is=TRUE) #lendo
arquivo
ako<-cbind(ako[,14:15],ako[,1:13],ako[,16:21])#é exigencia da funcao que as
col 1 e 2 sejam X e Y!
options(max.print=30000) #definindo o máximo de linhas q será mostrado dos
objetos...

  #Opção sem o cálculo do MPC
ako.mcpA<-selection.MCP(coords = ako, id = ako$Tag , session = ako$Session ,
minsession = 2 , mincapture = 5, calculation = FALSE)

  #Opção com o cálculo do MPC e com os pontos do MPC
  #capturas em no mínimo 2 sessões diferentes e pelo menos 5 pontos
diferentes.
ako.mcpB<-selection.MCP(coords = ako, id = ako$Tag , session = ako$Session ,
minsession = 2 , mincapture = 5, calculation = TRUE, percent=c(90,100),
unin=c("m"), unout=c("m2"), mcpshow=TRUE)

#2
  #criando um data.frame de acordo com exigências da função:
id1<-rep(1,10)
id2<-rep(2,7)
id3<-rep(3,8)
id4<-rep(4,3)
x1<-seq(1:10)
y1<-c(seq(2:8),rep(9,3))
x2<-c(rep(2,5),rep(3,2))
y2<-seq(10:3)
x3<-rep(2,8)
y3<-c(seq(4:9),1)
x4<-c(1,4,5)
y4<-c(3,7,8)
indiv<-as.data.frame(c(id1,id2,id3,id4))
coordx<-as.data.frame(c(x1,x2,x3,x4))
coordy<-as.data.frame(c(y1,y2,y3,y4))
teste<-cbind(coordx,coordy,indiv)
library(gpclib)
        #Opção com o cálculo do MPC e com os pontos do MPC  #capturas em no
mínimo 1 sessão e pelo menos 4 #pontos diferentes.
testeresulta<-selection.MCP(coords = teste, id = teste[,3] ,  session =
rep(2,28), minsession = 1 , mincapture = 4,calculation =TRUE,
percent=c(50,75,100), unin=c("m"), unout=c("m2"),mcpshow=TRUE)

#3
```

```
library(gpclib)
del<-read.table("tab0delomyscamila.csv",sep=";",head=TRUE, as.is=TRUE)
del<-cbind(del[,14:15],del[,1:13],del[,16:21])#é exigencia da funcao que as
col 1 e 2 sejam X e Y!
delomys<-selection.MCP(coords = del, id = del$Tag , session = del$Session ,
minsession = 2 , mincapture = 3,calculation =TRUE, percent=c(80,95,100),
unin=c("m"), unout=c("m2"))
```

Código da Função

```
selection.MCP <- function (coords, id, session, minsession, mincapture,
calculation = TRUE, percent=c(80,95,100), unin = c("m"), unout=c("m2"),
mcpshow=FALSE)
{
  ##Verificação de argumentos
  #Parar a função caso não tenha o argumento "coords" ou se este argumento
não é um data.frame:
  if(missing(coords))   stop("É obrigatória a inclusão das coordenadas")
  if(class(coords)!= "data.frame")   stop(" Argumento ´coords´ deve ser um
data.frame")
  #Parar a função caso não tenha o argumento "id":
  if(missing(id))    stop("Precisa de um identificador dos pontos, nem que
seja uma coluna com 1 valor só")
  if(length(id)!=nrow(coords)) stop("Argumentos ´coords´ e ´id´ devem ter o
mesmo comprimento") #checando se cada ponto tem um ID associado
  #Aviso de que, caso não esteja especificado o mínimo de capturas, será
utilizado mínimo 5.
  if(missing(mincapture))    {warning("Será utilizado o default: mínimo de 5
capturas para seleção dos dados")
                              mincapture = 5} #determinando que o mincapture
será 5 dentro do if anterior
  if (missing(session)) #Caso não tenha o argumento "session",
  {  if(!missing(minsession)) #e tenha o argumento "minsession",
  {stop("Se um mínimo de sessões foi especificado, indique as sessões de
captura")} #pare a função com esse aviso
  if(missing(minsession))  #caso não tenha o argumento "session" nem o "min
session,
  {warning("Foi considerado sessão de captura = 1 para todos os pontos e
´minsession´ = 1")
   coords$session<-c(rep(1,nrow(coords))) #crie uma nova coluna "session"
completa com 1
   minsession<-1}   # e assuma "minsession" igual a 1!
  }
  else
  { if(length(session)!=nrow(coords)) stop("´Coords´ e ´session´ devem ter o
mesmo comprimento") #checando se cada ponto tem uma sessão de captura
associada e pare caso não!
    coords$session <- session} # Se não estiver faltando o argumento
```

```
"session", inclua uma coluna no data.frame coords com o vetor indicado
  #PREPARANDO O TERRENO PARA A SELEÇÃO DE DADOS:
  coords$id2<-as.character(id)  ### duplicando coluna ÍD como caracter! Será
importante depois!
  ##removendo NAs:
  id<-id[!is.na(coords[,1])&!is.na(coords[,2])] #Retire as posições de id
que, no data.frame "coords" tem NA no mesmo valor de linha da coluna 1 ou
que tenha NA no mesmo valor de linha da coluna 2.
  coords<-coords[!is.na(coords[,1]),] #Verifica se tem valor faltante na
col1 do coords e remove a linha caso tenha   # como remove a linha, não
preciso me preocupar com a verificação do "session"!
  coords<-coords[!is.na(coords[,2]),] #Verifica se tem valor faltante na
col2 do coords e remove a linha caso tenha
  id<-as.factor(id) #transdormando id em fator
  coords$xy<-paste (coords[,1],coords[,2], sep = " ", collapse = NULL)
#criando uma coluna com XY concatenados que será importante para garantir
pontos de captura diferentes
  ####INÍCIO DA SELEÇÃO DOS DADOS:
  #1)Selecionando somente aqueles com min de pontos de captura diferentes
(argumento mincapture):
  arr<- tapply(coords$xy,id, FUN= "unique") #aplicando a função "unique" na
colunaXY para cada id para gerar uma lista de vetores com os pontos
diferentes de captura de cada id(cada elemento do vetor será o texto
concatenado das coordenadas xy do ponto)
  #For que faz com que fiquei na lista "arr" só aqueles vetores com pelo
menos comprimento "mincapture"!
  for(i in length(arr):1)             # do comprimento de "arr" até 1...
  { if (length(arr[[i]])<mincapture)  #Se o comprimento do vetor na posição
i da lista "arr" é menor que o valor do argumento "mincapture",
  {arr[i]<-NULL}                      #elimine a posição i da lista "arr"
  }
  #For para criar uma lista dos data.frames dos Ids que têm captura em
"mincapture" pontos diferentes. É importante notar que o for anterior nos dá
uma lista de vetores de pontos de captura e esse nos dará subsets do
data.frame "coords"
  lis<-list(NA)                    #Criando uma lista com 1 NA
  for(i in length(names(arr)):1) #Desde o comprimento do vetor com os nomes
da lista "arr" até 1,
  {lis[[i]]<-subset(coords,coords$id2==names(arr[i]))#colocando na lista
"lis" os subset do data.frame "coords" conforme aqueles ids que sao os nomes
dos obj da lista "arr'
  }
  #2)For que exclui aqueles ids que foram capturados em menos de
"minsession" sessões:
  for(i in length(lis):1)   # desde o final até o começo do comprimento da
lista lis
  {j<-lis[[i]]      # Dando o nome "j" ao data.frame de posição i na lista
lis
   if(length(unique(j$session))<minsession)  #Se o comprimento do vetor com
1x cada elemento da coluna session for menor que o valor do argumento
"minsession",
```

```
   {lis[[i]]<-NULL}         #elimine a posição i da lista "lis"
  }
  fin<-do.call(rbind, lis)  # Criando um data frame que junta todos os
data.frames da lista mas que o id não é uma coluna. #Por isso que eu
dupliquei o id lá no início!#
  if(calculation==FALSE)  # Opção para argumento "cálculo". Se falso,
  { #Arrumando data.frame fin para que ele fique com as mesmas colunas q a
pessoa inseriu:
    fin$xy<-NULL #tira a coluna "xy" que foi inserida para poder executar a
função
    fin$id2<-NULL #tira a coluna "id2" que foi inserida para poder executar
a função
    fin$session<-NULL #tira a coluna "session" que foi inserida para poder
executar a função
    cat("\n Resultado: data.frame com IDs selecionados para cálculo e seus
pontos de captura") #texto explicando o retorno da função nesse caso.
    return(fin)} #termine a função e retorne o objeto "fin", o data.frame
com os IDs selecionados.
  ####PREPARANDO O TERRENO PARA O CÁLCULO DA ÁREA DE VIDA:
  #Verificação:
  if (!require(gpclib))  # Se não tiver pacote "gpclib" instalado e ativo,
    stop("Precisa do pacote gpclib, não disponível para Windows") #pare a
função com essa mensage de erro.
  #Criando funções básicas necessárias (adaptadas do pacote adehabitat):
  if (mincapture<5) warning("A estimativa da área de vida é mais precisa com
5  capturas ou mais")
  #######função MCP, que determina quais pontos deve ser usados para gerar o
mínimo polígono convexo possível
  "mcp" <- function(xy, id, percent)
  {
    ## Verificações
    if (percent>100)  #Se o valor colocado no argumento "percent" for maior
que 100,
    { warning("O MCP será calculado usando todos os pontos de captura
(percent>100)") #aviso de que será calculado com percent=100.
      percent<-100      #atribuindo valor 100 ao argumento.
    }
    if (min(table(id))<3)
      stop("Pelo menos 3 pontos de captura são necessários para calcular o
MCP")
    #
    id<-factor(id) #convertendo id em fator
    ## Computando o centróide das capturas para cada id
    r<-split(xy, id) #quebre o data.frame em uma lista "r" de data.frames de
cada id
    est.cdg<-function(xy){ apply(xy, 2, mean)} #Criando uma função que
calcula a média de cada coluna de um data.frame.
    cdg<-lapply(r,est.cdg)  #Aplica a função recém criada em cada data.frame
da lista "r" e coloca o resultado na lista "cdg"
    levid<-levels(id)       #Cria um vetor com os níveis do fator id.
```

```
    ## Preparando para outputs
    X<-0 #Cria objeto X com 1 valor numérico 0
    Y<-0 #Cria objeto Y com 1 valor numérico 0
    ID<-"0" #Cria objeto ID com 1 caracter 0
    ## Para cada animal:
    for (i in 1:nlevels(id)) {  #de 1 até o número de níveis de id (para
cada id),
      df.t<-r[[levid[i]]]       #nomeia "df.t", o data.frame que está na
posição em questão levid[i] da lista "r".
      cdg.t<-cdg[[levid[i]]]    #nomeia "cdg.t" cada objeto que está na
posição em questão da lista "cdg", que tem as médias de x e y de cada id.
      ## Distancia das capturas até o centróide:
      dist.cdg<-function(xyt) { # criando uma função que calcula a distância
entre dois pontos
        d<-sqrt( ( (xyt[1]-cdg.t[1])^2 ) + ( (xyt[2]-cdg.t[2])^2 ) ) #
através do teorema de Pitágoras (x^2 + y^2 = h^2)
        return(d)               # retorna d, que é a hipotenusa (se for um
triângulo) e a distância entre os pontos.
      }
      di<-apply(df.t, 1, dist.cdg) # aplica a função recém criada nas linhas
dos data.frames da lista "r" e armazena o resultado no vetor "di".
      key<-c(1:length(di))         # cria um vetor de 1 até o comprimento do
vetor "di".
      #Próximos passos são para excluir os pontos de acordo com o "percent"
escolhido: exclui do mais distante para o mais próximo.
      acons<-key[di<=quantile(di,percent/100)] #Deixa no vetor "key" as
posições do vetor "di" que permanecem depois de excluir 100-"percent" dos
pontos. Chama de "acons" esse vetor com valores que indicam posições.
      xy.t<-df.t[acons,] #No data.frame da vez (ainda estamos dentro do
for), mantém as linhas que estão no vetor "acons" e chama de xy.t.
      ## Determinando as coordenadas que irão compor o MCP.
      #Função "chull()": Determina quais pontos dentro de um grupo, se
unidos, contém todos os outros pontos dentro do polígono formado.
      coords.t<-chull(xy.t[,1], xy.t[,2]) #Aplique na coluna 1 e na coluna 2
do data.frame xy.t a função chull(). Nomeia o conjunto de linhas resultantes
de coords.t
      xy.bord<-xy.t[coords.t,]  #Mantém, no data.frame "xy.bord" as linhas
do data.frame xy.t que correspondem aos valores do vetor coords.t!
      X<-c(X,xy.bord[,1])               #Coloca no objeto X a coluna com as
coordenadas x de cada ponto selecionado para o id em questão
      Y<-c(Y,xy.bord[,2])               #Coloca no objeto Y a coluna com as
coordenadas y de cada ponto selecionado
      ID<-c(ID, rep(as.character(levid[i]), nrow(xy.bord))) #Cria uma
repetição do valor que está na posição em questão do vetor levid (com os
níveis de id) com número de vezes igual ao número de linhas do data.frame
xy.bords.
    }
    ## Outputs:
    ID<-as.data.frame(ID)   #transforma conjunto ID em data.frame
    re<-cbind.data.frame(ID,X,Y)  #Junta os 3 objetos como data.frame no
objeto "re"
```

```
    re<-re[-1,]           #exclui a primeira linha que era valores e
caracteres zeros.
    re[,1]<-factor(re[,1])#transforma coluna 1(id) em fator.
    names(re) <- c("ID","X","Y") #nomeia as colunas de re
    return(re)            #define que o output dessa função é "re"
  }
  ##Criando função mcp.area, que calcula a área do polígono formado pelos
pontos selecionados em mcp()
  "mcp.area" <- function(xy, id, percent, unin=c("m", "km"),
                         unout=c("ha", "km2", "m2"))
  {
    ## Verificações
    unin<-match.arg(unin) #verifica se o valor colocado está dentre as
opções do argumento e atribui o valor ao objeto "unin"
    unout<-match.arg(unout)#verifica se o valor colocado está dentre as
opções do argumento e atribui o valor ao objeto "unout"
    ## Bases
    lev<-percent  #atribui nome lev ao conjunto percent para poder fazer
operações depois sem preder o valor de percent
    res<-list()   #cria uma lista vazia "res"
    ar<-matrix(0,nrow=length(lev),  #cria uma matrix de zeros com número de
colunas igual ao número de ids e número de
               ncol=nlevels(factor(id)))#linhas igual ao número de elementos
do conjunto "percent"
    lixy <- split(xy, id) #Quebra o data.frame xy para cada id em uma lista
"lixy"
    le <- names(lixy)     #cria um vetor com os nomes dos objetos
(data.frames) da lista "lixy". São os nomes dos ids.
    ## For que, para cada id (nivel do fator id), calcula o MCP e sua área:
    for (i in 1:length(lev)) {   #de 1 até comprimento do vetor
lev(porcentagens de pontos que ficam)
      ## Calculando MCP de cada id:
      ar[i,] <- unlist(lapply(lixy, function(z) {   #Em cada linha da matriz
"ar", depois de aplicar a função que está sendo criada em cada objeto da
lista "lixy" e reverter essa lista, coloque o resultado disso.
        res<-mcp(z, rep(1,nrow(z)), percent=lev[i]) #A função: Para cada
valor de porcentagem, calcula o MCP e armazena o valor numa lista "res".
        class(res)<-"data.frame"                    #Converte "res" em
data.frame
        return(area.poly(as(res[,2:3], "gpc.poly")))#Retorne a área do
poligono formada pelos pontos resultantes de MCP depois de transformar "res"
num objeto de classe "gpc.poly"
      }))
    }
    ar <- as.data.frame(ar)  #converte a matriz "ar" em data.frame
    names(ar)<-le            #e dá o nome das colunas com os níveis de "id".
    ## Definindo valores do output de acordo com a unidade do input e do
output (argumentos unin e unout)
    if (unin=="m") {   #Se input em metros
      if (unout=="ha") #e output em hectares,
```

```
        ar<-ar/10000   #divida todos os valores de "ar" por 10mil
      if (unout=="km2") #Se input em metros e output em kilometros
quadrados,
        ar<-ar/1000000  #divida todos os valores de "ar" por 1milhão
    }
    if (unin=="km") {  #Se input em kilometros
      if (unout=="ha") #e output em hectares,
        ar<-ar*100     #multiplique todos os valores de "ar" por 100
      if (unout=="m2") #Se input em kilometros e output em metros quadrados,
        ar<-ar*1000000 #multiplique todos os valores de "ar" por 1 milhão
    }
    ## output
    row.names(ar)<-lev #nomeie as linhas do data.frame com os valores do
conjunto de porcentagens.
    return(ar) #retorne o data.frame "ar"
  }
  #Depois de criar as funções, aplicando as duas:
  esti.area<-mcp.area(fin[,1:2], fin$id2, percent,unin,unout) #Aplicando
função mcp.area em cada id do objeto fin (data.frame de ids selecionados),
guardando resultado no objeto est.area.
  ##Extraindo os pontos que são os vértices do MCP (para caso a pessoa
esteja interessada em saber-argumento "mcpshow"=TRUE):
  lista.pontos<-list() #criando uma lista vazia que vai acomodar os
data.frames de cada valor de %.
  for(i in 1:length(percent)) #começando um for: de 1 a comprimento do
argumento "percent"
  { lista.pontos[[i]]<- mcp(fin[,1:2], fin$id2, percent[i])# Calcula o MCP
para cada valor de percent e armazene na posição igual a desse valor no arg.
percent
  }
  names(lista.pontos)<- percent #Nomeia os itens da lista com os valores do
conjunto percent
  ##Output final:
  #Limpando o objeto "fin" para que ele tenha as mesmas colunas do objeto
inserido na função:
  fin$xy<-NULL #tira a coluna "xy" que foi inserida para poder executar a
função
  fin$id2<-NULL #tira a coluna "id2" que foi inserida para poder executar a
função
  fin$session<-NULL #tira a coluna "session" que foi inserida para poder
executar a função
  if(mcpshow==TRUE)
  {cat("\n Lista com seguintes itens: \n Item 1 é o subset com os dados
selecionados \n Item 2 é a tabela com valor da estimativa MPC, na unidade
escolhida em ´unout´\n Item 3
   é uma lista de data.frames com os pontos de captura de cada id que compõe
os vértices dos polígonos utilizando cada valor de ´percent´") #Texto
explicando cada item da lista
   lista.final<-list(fin,esti.area,lista.pontos)#Criando uma lista para o
retorno com os data.frames subconjunto e de tamanhos da área de vida e a
lista com os pontos utilizados de acordo com a %depontos utilizada
```

```
   names(lista.final) <- c("subconjunto","TamanhoMCP","PontosUtilizados")
#Dando o nome de cada elemento da lista
  }
  else
  {cat("\n Lista com seguintes itens: \n Item 1 é o subset com os dados
selecionados \n Item 2 é a tabela com valor da estimativa MPC, na unidade
escolhida em ´unout´") #explicando, para quem rodar a função, o que são os
resultados.
   lista.final<-list(fin,esti.area) #Criando uma lista para o retorno com o
data.frame subconjunto e os tamanhos da área de vida
   names(lista.final) <- c("subconjunto","TamanhoMCP") #Dando o nome de cada
elemento da lista
  }
  return(lista.final) #Definindo que o retorno será uma lista que varia seu
conteúdo de acordo com o último "if".
}
#Acabou já! =)
```

Arquivos Para Teste

tabela akodon

tabela delomys

Arquivos da Função

código da função

help da função

help em .pdf

From:
http://labtrop.ib.usp.br/ - **Laboratório de Ecologia de Florestas Tropicais**

Permanent link:
http://labtrop.ib.usp.br/doku.php?id=cursos:ecor:05_curso_antigo:r2015:alunos:trabalho_final:gabriela.marin:start

Last update: **2020/07/27 18:48**