Joana Carvalhaes Borba de Araujo

joana_araujo@usp.br

Mestranda em Biologia Comparada, Faculdade de Filosofia, Ciências e Letras de Ribeirão Preto, USP

Tenho interesse por pesquisa em conservação de aves ameaçadas de extinção e atualmente desenvolvo o projeto “Tamanho populacional, razão sexual e ocupação da paisagem de uma população de mutum-do-sudeste (*Crax blumenbachii* Spix, 1825) reintroduzida na RPPN Fazenda Macedônia, Ipaba, MG”, sob orientação do Prof. Dr. Adriano Chiarello, do Laboratório de Ecologia e Conservação de Ribeirão Preto, e colaboração do Prof. Dr. Luís Fábio Silveira, do Museu de Zoologia da USP. Currículo Lattes

Exercícios

exec

Trabalho final

Plano A

Em alguns métodos de pesquisa em campo, é fundamental que o pesquisador saiba as distâncias reais percorridas durante a amostragem. No caso de *distance sampling* por transecção linear, por exemplo, essa informação é necessária tanto para o cálculo da probabilidade de cobertura das transecções, quanto da probabilidade de detecção, que são os parâmetros base para as estimativas de tamanho populacional da área de estudo. Entretanto, em muitas áreas de relevo acidentado, tais distâncias podem ser subestimadas por mapas em que as curvas de nível são ausentes ou não representam com fidelidade a situação da área, levando a uma superestimação da densidade populacional. Diante deste problema, pretendo desenvolver uma função que permita estimar estas distâncias através de pontos coletados por GPS.

Dados de entrada: A função aceitará como argumento um data frame de 4 colunas: identificação dos pontos e seus respectivos valores de latitude, longitude (UTM) e altitude (m).

A lógica: Em um primeiro momento, a função deverá gerar uma regressão linear entre os dados de latitude e longitude que será equivalente ao trajeto linear percorrido. Os resíduos, interpretados como erros do GPS, serão projetados nesta linha a partir dos valores dos coeficientes, gerando uma nova sequência de pontos alinhados com coordenadas x,y. A seguir, a função irá calcular a distância entre cada ponto por simples trigonometria, a partir dos deltas x e y. O mesmo princípio será aplicado para o cálculo da distância real percorrida, que será nada menos que a soma das hipotenusa, sendo os catetos a “distância percorrida” e o “delta h” entre cada ponto da sequência.

Dados de saída: Como resultado, a função deverá retornar o comprimento da transecção em metros e um gráfico que represente o perfil altimétrico aproximado da transecção. Naturalmente, quanto mais pontos por distância percorrida houver nos dados de entrada, mais refinados serão os resultados.

Alternativas: Adicionalmente, posso acrescentar um argumento para determinar se o trajeto de interesse é mesmo linear. Caso contrário, a função executaria os mesmos comandos, entretanto, sem a etapa de projetar os dados para uma regressão linear. Outra opção seria incluir como argumento a possibilidade de eliminar da análise os dados que se afastem muito da linha estabelecida pela regressão. Neste caso, o valor limite para o distanciamento dos resíduos seria determinado pelo próprio usuário e a função retornaria um aviso de que o corte ocorreu.

> Acho que a proposta A é viável e bacana. Só não entendi direito como vai lidar com a conversão dos dados de coordenadas geográficas para cartesianas simples. Não precisa de algum ajuste para a correção da projeção? Talvez na escala de trabalho isso não seja importante... de qq forma confirme! — *Alexandre Adalardo de Oliveira 2014/04/14 20:05*
>
> ---
>
> também gostei da proposta, acho interessante e factível, mas não entendi porque assumir erro do GPS. Veja neste vídeo como é difícil traçar uma linha reta no olhometro Linque Não seria melhor trabalhar com as coordenadas reais obtidas, e calcular a distância “real” entre pontos? Ao plotar tudo numa linha só, vc está perdendo informação de distância, e dependendo do seu relevo e de outros fatores, a quantidade de informação perdida pode ser semelhante à perdida utilizando um mapa ruim. Eu assumiria a possibilidade de não-linearidade e calcularia a distância total —- *Vitor Rios*
>
> ---
>
> Achei sua proposta A bem legal. Não sei se essa regressão que você propõe é um procedimento válido, mas se for um procedimento usual você pode fazer isso mesmo. Aparentemente, você entende o algoritmo envolvido e tem clareza dos parâmetros da sua função e do que ela retorna. E isso é muito bom!
>
> Se der tempo de fazer o gráfico, faça! Sobre a possibilidade de eliminar valores “outliers”: pense bem em como isso vai ser definido, pode ser uma boa alternativa mas tem que estar bem explicado no help da função como isso funciona.
>
> A proposta B está bem simples, então acho que você pode

seguir com a A mesmo. —- *Danilo G. Muniz*

Proposta ajustada

Agradeço as sugestões de todos e peço desculpas pela demora eu retorna-las, pois estava em campo! Alexandre, no caso eu vou utilizar os dados de coordenadas em UTM dentro de uma mesma quadrícula geográfica, o que já é similar a um plano cartesiano comum, e presumindo que os ajustes de projeção já estão incorporados aos dados de acordo com o DATUM em que o pesquisador está trabalhando. A limitação, entretanto, é que a função não servirá para áreas presentes em mais de uma quadrícula. Vitor, a ideia de assumir erro do GPS vem da minha própria experiência em campo, pois no interior da mata ele realmente gera erros enormes! É claro que o transecto nunca será perfeitamente retilíneo, mas a análise em *distance sampling* despreza essas pequenas distorções e trabalha com o pressuposto de uma reta conforme o desenho amostral, pois caso contrário, fica impossível extrapolar os dados para toda a área de estudo. Danilo, aprimorei a minha ideia de forma que a reta considerada agora será aquela que liga o ponto inicial e final do transecto, e não mais a regressão entre todos os pontos. Acho que assim faz mais sentido!

Plano B

Em Ecologia, uma das principais premissas da pesquisa em campo é a aleatoriedade na coleta dos dados. Pensando nisso, proponho criar uma função que sorteie a ordem de amostragem entre pontos para trabalhos de *distance sampling* ou qualquer outro método que dependa da aleatorização de unidades amostrais.

Dados de entrada: Três vetores com as seguintes especificações: o primeiro contendo a identificação dos pontos e mais dois com as coordenadas em UTM.

A lógica: Se rodada no modo *default*, a função nada mais será que um equivalente à função sample( ) sem reposição. O diferencial aqui está na possibilidade de direcionar o sorteio de modo mais viável logisticamente. Minha ideia é acrescentar argumentos que determinem quantos períodos amostrais estão previstos no projeto, a duração de cada período, a velocidade de deslocamento do pesquisador entre as unidades amostrais e o tempo de amostragem por ponto. Baseado nestes argumentos, a função será capaz de calcular quantos pontos o pesquisador é capaz de fazer por período amostral e, então, ordenar a amostragem de forma otimizada, agrupando pontos mais próximos (até uma distância determinada pelo usuário) em um mesmo período.

Dados de saída: Um vetor com a ordem prevista de amostragem organizado por período amostral na forma de um array de dimensões 1,n,z; sendo n = número de unidades amostradas/período de amostragem e z = quantidade de períodos amostrais prevista no projeto.

Página de ajuda

```
altimetric                package:unknown                R Documentation

            Perfil altimétrico em transecções retilíneas e trajetos não-
lineares

Description:
altimetric calcula distâncias percorridas em trajetos lineares ou não,
considerando variações altimétricas. A função ainda apresenta graficamente o
deslocamento horizontal e o perfil altimétrico da transecção.

Usage:
altimetric(lat,long,alt,out=Inf,linear=TRUE)

Arguments:
  lat  : vetor numérico. Coordenadas de latitude em sistema UTM
  long  : vetor numérico. Coordenadas de longitude em sistema UTM
  alt  : vetor numérico. Valores de altitude (m)
  out  : numérico. Valor limite de desvio do trajeto a partir do qual, os
dados são descartados
  linear  : lógico. Se TRUE os pontos são corrigidos para uma reta, se FALSE
são utilizadas as coordenadas originais.

Details:
altimetric funciona apenas para trajetos lineares inseridos dentro de uma
mesma zona geográfica conforme sistema UTM.
A função compreende o primeiro e o último valor de cada vetor como extremos
do trajeto, eliminando todos os pontos que ultrapassem estes limites.

Value:
altimetric retorna ao usuário valores numéricos e figuras gráficas:
  Distância horizontal (m) : distância percorrida desconsiderando a altitude
  Distância percorrida (m) : distância percorrida considerando variações
altimétricas
  Deslocamento horizontal  : gráfico do deslocamento superficial
  Perfil altimétrico  : gráfico do deslocamento topográfico

Author:
Joana Carvalhaes Borba de Araujo
joana_araujo@usp.br

Examples:

#transecto linear
lat1 =
c(7856507,7856712,7856695,7856780,7856773,7856810,7856834,7856670,7856652,78
56640,7856630,7856645,7856616,7856613,7856624,7856603,7856601,7856573,785656
```

```
3,7856561,7856540,7856520,7856523,7856528,7856538,7856499,7856504,7856525,78
56617,7856868,7856885,7856930,7856938,7856931,7856959,7856958,7856976,785698
2,7857009,7856989,7856947,7856744,7856831,7856855,7856909,7856994)
long1 =
c(773012.3,773617.1,773506.1,773739.9,773744.4,773826.2,773863.9,773455.3,77
3417.9,773414.2,773411.8,773374.4,773319.0,773323.6,773324.6,773280.9,773263
.3,773192.2,773182.4,773175.6,773106.1,773038.1,773047.7,773063.6,773044.3,7
73000.1,772980.4,773175.2,773230.9,774028.9,774026.6,774122.1,774188.8,77417
9.4,774231.6,774242.7,774274.9,774330.0,774334.1,774345.2,774515.8,774038.1,
773546.3,773507.5,773340.4,774337.3)
alt1 =
c(230,270,235,300,300,315,305,230,200,210,220,220,270,270,260,300,330,300,26
0,245,220,260,265,250,250,230,230,235,310,270,265,300,250,220,215,225,210,23
0,230,230,220,290,330,340,305,230)
altimetric(lat1,long1,alt1,out=30)

#trajeto não linear
lat2=c(7853029,7853075,7853108,7853045,7853078,7852996,7852713,7852615,78525
61,7852550)
long2=c(766121.2,766298.6,766592.3,766679.6,766957.5,767224.8,767137.5,76730
6.9,767343.9,767563.6)
alt2=c(230,210,350,340,260,200,250,340,270,220)
altimetric(lat2,long2,alt2,linear=FALSE)
```

Código da função

```
altimetric <- function(lat, long, alt, out=Inf, linear=TRUE)
    {
    n = length(lat) #número de pontos nos dados de entrada
    x = c(long[1],long[n]) #determina longitude inicial e final
    y = c(lat[1],lat[n]) #determina latitude inicial e final
    x11() #abre uma janela gráfica
    par(mfrow=c(2,1)) #divide a janela gráfica em duas linhas
    plot(lat~long,pch="",xlab="Longitudes (UTM)",ylab="Latitudes (UTM)",main
= "Deslocamento horizontal",xaxs="r",yaxs="r") #gera gráfico latlong vazio
    rect(par("usr")[1], par("usr")[3], par("usr")[2],
par("usr")[4],col="khaki2") #colore o fundo do gráfico representando a
superfície vista por cima
    points(long,lat,pch=20) #plota os pontos do GPS
    if(linear==TRUE) #se o trajeto considerado é linear...
        {
        segments(x[1],y[1],x[2],y[2]) #plota a reta do transecto entre o
ponto final e o inicial
        if(y[1]==y[2]) #se o transecto mantem latitude constante (sentido
leste-oeste)...
            {
            lat3 = rep(y[1],n) #projeta todos os pontos na latitude dos
extremos
            long3 = long #mantem long inicial
```

```
            desvios = abs(lat3-lat) #calcula quanto o ponto se afasta da
reta
            dados = data.frame(lat3,long3,alt,lat,long,desvios) #une os
dados em um data frame
            fora = dados[long3<min(x)|long3>max(x),] #determina quais os
pontos que estão antes ou depois dos extremos do transecto
            dados1 = dados[long3>=min(x)&long3<=max(x),] #deixa apenas os
pontos entre os extremos do transecto
            dados2 = dados1[order(dados1$long3,na.last=NA),] #ordena os
pontos em valores crescentes de long
            }
        if(x[1]==x[2]) #se o transecto mantem longitude constante (sentido
norte-sul)...
            {
            lat3=lat #mantem lat inicial
            long3=rep(x[1],n) #projeta todos os pontos na longitude dos
extremos
            desvios = abs(long3-long) #calcula quanto o ponto se afasta da
reta
            dados = data.frame(lat3,long3,alt,lat,long,desvios) #une os
dados em um data frame
            fora = dados[lat3<min(y)|lat3>max(y),] #determina quais os
pontos que estão antes ou depois dos extremos do transecto
            dados1 = dados[lat3>=min(y)&lat3<=max(y),] #deixa apenas os
pontos entre os extremos do transecto
            dados2 = dados1[order(dados1$lat3,na.last=NA),] #ordena os
pontos em valores crescentes de lat
            }
        if(y[1]!=y[2]&x[1]!=x[2]) #se a reta varia tanto em latitude, quanto
em longitude...
            {
            transecto = lm(y~x) #determina a regressão linear entre os
pontos inicial e final
            a = transecto$coefficients[1] #valor do intercepto
            b = transecto$coefficients[2] #valor do coeficiente de
inclinação
            lat2 = c(y[1],a + b*long[seq(2,n-1)],y[2]) #valores de lat que
projetam os pontos na reta, perpendicularmente ao eixo y
            long2 = c(x[1],(lat[seq(2,n-1)]-a)/b,x[2]) #valores de long que
projetam os pontos na reta, perpendicularmente ao eixo x
            lat3 = (lat+lat2)/2 #valores de lat que projetam os pontos na
reta, perpendicularmente à reta
            long3 = (long+long2)/2 #valores de long que projetam os pontos
na reta, perpendicularmente à reta. Obs: lm(y~x) = lm(lat3~long3).
            delta.lat = abs(lat-lat3) #delta y entre a posição original dos
pontos e a posição projetada no transecto
            delta.long = abs(long-long3) #delta x entre a posição original
dos pontos e a posição projetada no transecto
            desvios = round(sqrt(delta.lat^2 + delta.long^2)) #desvio do
ponto em relação ao transecto
```

```
            dados = data.frame(lat3,long3,alt,lat,long,desvios) #une os
dados em um data frame
            fora = dados[long3<min(x)|long3>max(x),] #determina quais os
pontos que estão antes ou depois dos extremos do transecto
            dados1 = dados[long3>=min(x)&long3<=max(x),] #deixa apenas os
pontos entre os extremos do transecto
            dados2 = dados1[order(dados1$long3,na.last=NA),] #ordena os
pontos em valores crescentes de long
            }
points(dados2$long[dados2$desvios>=out],dados2$lat[dados2$desvios>=out],pch=
20,col=2) #apresenta em vermelho os pontos com desvio maior do que out
        points(fora$long,fora$lat,pch=20,col="blue") #apresenta em azul os
pontos além das extremidades
        n.fora = length(fora[,1]) #calcula quantos pontos estão além das
extremidades do transecto
        outliers = sum(dados2$desvios>out) #determina quantos pontos entre
os extremos do transecto têm desvio maior que out
        legend("bottomright",c("além da extremidade",paste("desvio
>=",out,"m",sep=" ")),bty="n",pch=20,col=c("blue",2),cex=0.9,pt.cex=1)
#plota a legenda dos pontos eliminados
        cat(outliers+n.fora, "outliers encontrados:\n",n.fora,"pontos
ultrapassam as extremidades do transecto - excluídos\n") #informa quantos
pontos estão além da faixa válida e quantos foram excluídos por
ultrapassarem os extremos
        LAT = dados2$lat3[dados2$desvios<out] #cria vetor com dados de lat
dos pontos que não ultrapassam out
        LONG = dados2$long3[dados2$desvios<out] #cria vetor com dados de
long dos pontos que não ultrapassam out
        ALT = dados2$alt[dados2$desvios<out] #cria vetor com dados de alt
dos pontos que não ultrapassam out
        if(out!=Inf)#se o usuário determinou uma faixa limite de desvio
            {
            cat(outliers, "pontos a partir de",out,"m do transecto -
excluídos\n") #informa quantos pontos foram excluídos por ultrapassar out
            }
        }
    if(linear==FALSE) #se o trajeto em questão não é linear...
        {
        LAT=lat #latitudes se mantêm iguais aos pontos originais
        LONG=long #longitudes se mantêm iguais aos pontos originais
        ALT=alt #altitudes se mantêm iguais aos pontos originais
        segments(long[1:n],lat[1:n],long[1:n+1],lat[1:n+1],lty=3) #plota a
trajetória percorrida
        }
    n2=length(LAT) #determina o número de pontos remanescentes
    delta.LAT = abs(LAT[2:n2]-LAT[1:n2-1]) #calcula variação em lat entre
pontos em subsequentes
    delta.LONG = abs(LONG[2:n2]-LONG[1:n2-1]) #calcula variação em long
entre pontos subsequentes
    dist.horizontal = sqrt(delta.LAT^2 + delta.LONG^2) #calcula distância
horizontal entre os pontos
```

```
    total.horizontal = sum(dist.horizontal) #calcula a distância horizontal
total do transecto (desconsiderando alt)
    delta.ALT = abs(ALT[2:n2]-ALT[1:n2-1]) #calcula variação em alt entre
pontos subsequentes
    dist.real = sqrt(dist.horizontal^2 + delta.ALT^2) #calcula distância
percorrida entre os pontos considerando a variação da altitude
    total.percorrido = sum(dist.real) #soma as distâncias percorridas em
todos os trechos
    (dist.percorrida = rep(NA,n2-1)) #cria um vetor vazio para as distâncias
horizontais
    for(i in 1:n2-1) #considerando o número de intervalos entre pontos...
        {
        dist.percorrida[i] = sum(dist.horizontal[1:i]) #preenche o vetor com
a soma da distância horizontal percorrida até cada ponto
        }
    dist.percorrida2 = c(0,dist.percorrida) #cria um novo vetor igual ao
anterior, mas iniciando em zero
    plot(dist.percorrida2,ALT,pch="",xlab="Distância horizontal
(m)",ylab="Altitude (m)", main = "Perfil
altimétrico",ylim=c(0,max(ALT)+max(ALT)/2),xaxs="i",yaxs="i") #gera um
perfil altimétrico vazio
    rect(par("usr")[1], par("usr")[3], par("usr")[2], par("usr")[4], col =
"lightblue") #colore o fundo em cor que representa o céu
    polygon(c(0,dist.percorrida2,dist.percorrida2[n2]),c(0,ALT,0),
col="khaki2") #plota o perfil altimétrico do transecto em cor que representa
a superfície terrestre
    resulta =list(total.horizontal,total.percorrido) #cria lista com os
resultados de interesse
    names(resulta) = c("Distância horizontal (m)","Distância percorrida
(m)") #nomeia os resultados
    return(resulta) #determina o que a função irá retornar
    }
```

Arquivo da função

altimetric