Milene Gomes Jannetti

Mestranda em Cronobiologia, Departamento de Fisiologia, Instituto de Biociências, USP

Meus Exercícios

- Exercício 1
- Exercício 4
- Exercício 5
- Exercício 6.2
- Exercício 7
- Exercício 8
- Exercício 9.2

Trabalho Final

Propostas

Minhas propostas

A função rota (Proposta B)

- Arquivo da função
- Página de ajuda

O código

```
 rota<- function(posx, posy, t.partida=1, vel=5, p.partida, p.destino,
p.parada=NULL, ordem.fixa=TRUE, encurtar="tempo.espera", tx.amostragem=1,
hora1=0, dia1=0, xlab="Unidade de medida", ylab="Unidade de medida",
main="Mapa do espaço")
{
##Checar se a primeira coluna dos data.frames posx e posy são caracteres (se
podem ser usados como os nomes dos pontos)
if(class(posx[,1])!="character"|class(posy[,1])!="character")
{
##Caso a classe dessa coluna não seja character, um objeto será criado com
uma mensagem para o usuário
format<-"Primeira coluna dos data.frames posx e posy deve ser da classe
character"
##Retornar aviso para usuário e encerrar os comandos
```

```
return(format)
}
##Verificar se existem paradas
if(length(p.parada)==0)
    {
    ##Caso não existam paradas, será criada uma matriz de 1 linha e 1
coluna, para ser usada posteriormente em uma função for
    pslist=matrix(0)
    }
##Criar condição para o caso de existirem paradas
else
    {
    ##Existindo paradas, verificar se a ordem delas será fixa ou não
    if(ordem.fixa==FALSE)
        {
        ##Se ordem.fixa=FALSE, então será necessário avaliar todas as
possibilidades de sequência de paradas.
        ##Criar um data frame contendo, em cada coluna, o vetor dos pontos
de paradas, com comprimento x, repetido x vezes
        dframe<-data.frame(matrix(rep(p.parada,length(p.parada)),
nrow=length(p.parada)), stringsAsFactors=FALSE)
        ##Obter todas as combinações possíveis de sequência de paradas. A
função expand.grid cria sequências com repetições.
        cjuntos<-expand.grid(dframe, stringsAsFactors=FALSE)
        ##Selecionar só as combinações sem repetições, ou seja, para as
quais cada ponto apareça só uma vez.
        ##Contar quantas vezes cada ponto aparece em cada combinação
        count<-apply(cjuntos,1,table)
        ##Criar vetor resposta com o comprimento do data.frame das
combinações
        qual.usar<-rep(NA, length(cjuntos[,1]))
        ##Analisar cada uma das combinações
        for(i in 1:length(cjuntos[,1]))
            {
            ##No vetor resposta, armazenar um valor lógico para selecionar
as contagens que contém todos os elementos (ou seja, cujo comprimento seja
igual ao de p.parada)
            qual.usar[i]<-length(count[[i]])==length(p.parada)
            }
        ##Criar data.frame unindo as combinações possíveis com o vetor
lógico
        aval<-data.frame(cjuntos,qual.usar,stringsAsFactors=FALSE)
        ##Selecionar as combinações para as quais o vetor lógico seja TRUE
(contagens que contém todos os elementos)
        validos<-subset(aval,subset=(aval$qual.usar==TRUE))
        ##A matriz a ser utilizada conterá, em cada linha, uma sequência de
pontos de parada
        pslist<-as.matrix(validos[,1:length(p.parada)])
        }
    ##Criar condição para o caso da ordem dos pontos de parada ser fixa
```

```
    if(ordem.fixa==TRUE)
        {
        ##A matriz utilizada terá apenas uma linha, que será a sequência de
pontos determinada em p.parada
        pslist<-matrix(p.parada, nrow=1)
        }
    }
##Criar uma lista para guardar todas as possibilidades de caminho a serem
calculadas, com cada sequência de paradas
melhor.ordem<-vector("list",length(pslist[,1]))
##Criar um vetor para armazenar as distâncias ou horários de chegada, a
depender do critério escolhido em "encurtar", e, posteriormente, calcular
uma pontuação para cada rota
chegada<-rep(NA,length(pslist[,1]))

#### Construindo cada rota ####

##Calcular uma rota para cada linha da matriz pslist
for(y in 1:length(pslist[,1]))
    {
    ##Transformar a linha y da matriz em character
    pontos<-as.character(pslist[y,])
    ##Criar o vetor final de caracteres, contendo os pontos de partida e
destino
    if(length(p.parada)==0)
        {
        ##Caso não haja pontos de parada, desconsiderar pslist
        ps=c(p.partida,p.destino)
        }
    ##Caso haja pontos de parada,...
    else
        {
        ##Unir os pontos de parada com a partida e o destino
        ps=c(p.partida,pontos,p.destino)
        }
    ##Criar um data frame com as informações que serão obtidas para cada
trecho da viagem
    resumo.viagem<-data.frame(
                ##Nome dos pontos, determinados pelo usuário
                "Nome do ponto"=ps,
                ##Posição dos pontos em X no horário de chegada. Por
enquanto só sabemos a posição do ponto de partida
                "Posição em
x"=c(posx[posx[,1]==as.character(ps[1]),(t.partida+1)],rep(NA,
(length(ps)-1))),
                ##Posição dos pontos em Y no horário de chegada. Por
enquanto só sabemos a posição do ponto de partida
                "Posição em
y"=c(posy[posy[,1]==as.character(ps[1]),(t.partida+1)],rep(NA,
(length(ps)-1))),
                ##Horário de chegada a ser calculado para cada trecho. Para
```

```
o ponto de partida, horário de chegada será marcado como t.partida
                "Horario de chegada"=c(t.partida,rep(NA, (length(ps)-1))),
                ##Tempo de espera entre a chegada do sujeito na posição e a
chegada do ponto nessa posição. No ponto de partida não há espera.
                "Tempo de espera"=c(0,rep(NA, (length(ps)-1))),
                ##Dia em que se chegará nessa posição. O primeiro dia será
marcado como 0.
                "Dia de chegada"=c(0,rep(NA, (length(ps)-1)))
                )
    ##Criar um vetor para guardar as distâncias percorridas em cada trecho,
para ser utilizado na pontuação de cada rota (vetor chegada)
    dist<-rep(NA,(length(ps)-1))

    #### Construindo cada trecho ####

    ##Obter as informações especificadas no data frame resumo.viagem para
cada trecho.
    for (x in 1:(length(ps)-1))
        {
        ##Criar vetor resposta das distâncias desde p.partida em t.partida
até o próximo ponto em diferentes horários, a partir de t.partida
        result<-
data.frame(horario=seq(from=(resumo.viagem[x,4]),to=(length(posx[1,])-1)),
distancia=rep(NA,(length(posx[1,])-resumo.viagem[x,4])))
            ##Calcular as distâncias para cada horário
            for (i in 1:(length(posx[1,])-resumo.viagem[x,4]))
            {
            ##Distância no eixo x para todos os horários
            distx<-posx[posx[,1]==ps[x],(resumo.viagem[x,4]+1)]-
posx[posx[,1]==ps[x+1],(resumo.viagem[x,4]+i)]
            ##Distância no eixo y para todos os horários
            disty<-posy[posy[,1]==ps[x],(resumo.viagem[x,4]+1)]-
posy[posy[,1]==ps[x+1],(resumo.viagem[x,4]+i)]
            ##Distância resultante, armazenada no vetor resposta
            result[i,2]<-sqrt(distx^2+disty^2)
            }
        #### Escolhendo o melhor caminho ####

        ##Calcular quanto tempo o sujeito levará para percorrer cada uma das
distâncias
        hora.alcance<-result[,2]/vel
        ##Calcular quanto tempo o sujeito ficará esperando na posição em que
chegou (tempo de espera) até que o ponto apareça na posição:
        ##(tempo gasto para o ponto chegar = intervalo de tempo desde a
coluna de partida até a coluna de chegada) menos ( tempo gasto para o
sujeito chegar)
        tempo.espera<-(result[,1]-resumo.viagem[x,4])-hora.alcance
        ##Se positivo, o sujeito está adiantado em relação ao ponto. Se
negativo, o sujeito está atrasado.
        ##Criar vetor lógico para verificar se tempo.espera é positivo.
```

```
        alcance<-(tempo.espera>=0)
        ##Criar data frame com os cálculos acima, feitos para cada horário
        caminho<-data.frame(result,hora.alcance,tempo.espera,alcance)
        ##Criar subset só com tempo.espera positivos, eliminando os atrasos.
        cam.possivel<-subset(caminho, subset=caminho$alcance=="TRUE")
        ##Anunciar erro caso nenhum caminho tenha tempo.espera positivo
        if(length(cam.possivel[,1])==0)
            {
            ##Pedir uma velocidade maior ou mais horários a partir de
t.partida
            error="Aumentar a velocidade ou obter a posição dos pontos em
mais horários"
            ##Retornar a mensagem de erro e encerrar os comandos
            return(error)
            }
        ##Se existirem caminhos com tempo.espera positivos, continuamos os
comandos
        else
            {
            ##A partir deste subset, o horário de chegada pode ser aquele
com menor tempo de espera ou menor distância.
            resumo.viagem[(x+1),4]<-switch(encurtar,
distancia=cam.possivel[cam.possivel$distancia==min(cam.possivel$distancia),1
],
tempo.espera=cam.possivel[cam.possivel$tempo.espera==min(cam.possivel$tempo.
espera),1])
            #### Armazenando os trechos ####

            ##Guardar a posição no eixo X do ponto em que chegamos, no tempo
em que ele aparecerá.
            resumo.viagem[(x+1),2]<-
posx[posx[,1]==as.character(resumo.viagem[(x+1),1]),(resumo.viagem[(x+1),4]+
1)]
            ##Guardar a posição no eixo Y do ponto em que chegamos, no tempo
em que ele aparecerá.
            resumo.viagem[(x+1),3]<-
posy[posy[,1]==as.character(resumo.viagem[(x+1),1]),(resumo.viagem[(x+1),4]+
1)]
            ##Guardar o tempo de espera que teremos
            resumo.viagem[(x+1),5]<-
cam.possivel[cam.possivel$horario==resumo.viagem[(x+1),4],4]
            ##Guardar a distância a ser percorrida neste trecho
            dist[x]<-
cam.possivel[cam.possivel$horario==resumo.viagem[(x+1),4],2]
            }
        }
        #### Armazenando e escolhendo as rotas ####

        ##Guardar o resumo.viagem de cada rota, para cada sequência de
pontos de parada
        melhor.ordem[[y]]<-resumo.viagem
```

```
        ##Guardar a pontuação de cada sequência, a depender do critério
escolhido em "encurtar"
        chegada[y]<-switch(encurtar,
            ##Se encurtar=distancia, a pontuação será a soma das distâncias
de cada trecho da rota
            distancia=sum(dist),
            ##Se encurtar=tempo.espera, a pontuação será o horário de
chegada no ponto de destino e a soma dos tempos de espera em cada trecho
tempo.espera=resumo.viagem[length(resumo.viagem[,1]),4]+sum(resumo.viagem[,5
]))
    }
    ##A rota escolhida será aquela que obtiver menor pontuação, independente
do critério escolhido
    chegada.final<-which(chegada==min(chegada))

    #### Retornando o data.frame e o mapa ####

    ##No caso de haver mais de uma rota com pontuação mínima, garantir que
todas sejam exibidas.
    ##Criar uma lista resposta para armazenar o data.frame de cada caminho
    resumo.viagem.final<-vector("list",length(chegada.final))
    ##Criar gráfico e armazenar data.frame para cada rota
    for(a in 1:length(chegada.final))
        {
        ##Selecionar uma das rotas da lista que tenha pontuação mínima
        res.viagem.final<-melhor.ordem[[chegada.final[a]]]
        ### Ajustar horário e dia de chegada em cada trecho de acordo com a
taxa de amostragem, hora1 e dia1 ###
        ##Calcular o ajuste para que a viagem inicie na hora1
        ajuste.inicio<-hora1-res.viagem.final[1,4]*tx.amostragem
        ##Atribuir os novos horários com o ajuste calculado
        res.viagem.final[,4]<-
res.viagem.final[,4]*tx.amostragem+ajuste.inicio
        ##Calcular o dia correspondente a cada horário de chegada ajustado
        res.viagem.final[,6]<-floor(res.viagem.final[,4]/24)
        ##Novo ajuste dos horários para limitá-los de 0 a 23 horas
        res.viagem.final[,4]<-res.viagem.final[,4]-res.viagem.final[,6]*24
        ##Ajuste dos dias de acordo com o valor determinado em dia1
        res.viagem.final[,6]<-res.viagem.final[,6]+dia1
        ### Ajustar tempo de espera de acordo com a taxa de amostragem
        res.viagem.final[,5]<-res.viagem.final[,5]*tx.amostragem
        ### Plot ###
        ##Abrir nova janela gráfica
        X11()
        ##Plotar todos os pontos da rota, no mapa xy
        plot(res.viagem.final[,2],res.viagem.final[,3],
xlim=c(min(posx[,2:(length(posx))]),max(posx[,2:(length(posx))])),
ylim=c(min(posy[,2:(length(posy))]),max(posy[,2:(length(posy))])),
xlab=xlab, ylab=ylab, main=(c(a,main)))
        ##Criar uma linha para conectar cada trecho da viagem
```

```
        lines(res.viagem.final[,2],res.viagem.final[,3])
        ##Colocar nomes correspondentes aos pontos
text(x=res.viagem.final[,2],y=res.viagem.final[,3],labels=res.viagem.final[,
1], pos=1,offset=0.5)
        ##Colocar o horário e dia correspondente a cada ponto
        ##Criar uma matriz com o horário e o dia de chegada em cada trecho
        hora.dia<-matrix(c(res.viagem.final[,4],res.viagem.final[,6]),
nrow=length(res.viagem.final[,6]))
        ##Criar uma função para unir os elementos em um único character
        juntar<-function(x){paste(x,collapse="; dia ")}
        ##Plotar o horário e dia de cada ponto, utilizando a função apply no
argumento labels
text(x=res.viagem.final[,2],y=res.viagem.final[,3],labels=apply(hora.dia,1,j
untar), pos=1,offset=1.5)
        ##Destacar os pontos de chegada e destino
points(res.viagem.final[c(1,(length(res.viagem.final[,2]))),2],res.viagem.fi
nal[c(1,(length(res.viagem.final[,2]))),3], pch=c(15,17), col=c(3,2),
cex=1.25)
        ##Permitir adição de elementos fora da área de plotting
        par(xpd=TRUE)
        ##Adicionar legenda para os pontos de partida e destino
legend(max(posx[,2:(length(posx))])/1.25,max(posy[,2:(length(posy))])/0.8,c(
"Partida", "Destino"), pch=c(15,17), col=c(3,2))
        ##Restabelecer a condição padrão
        par(xpd=FALSE)
        ##Armazenar o data.frame na lista resposta
        resumo.viagem.final[[a]]<-res.viagem.final
        }
    ##Retornar o resumo da(s) rota(s)
    return(resumo.viagem.final)
}
```

Ajuda

```
rota                package:unknown                R Documentation

Cálculo do caminho de um ponto até outro, com mudança de posição dos pontos
ao longo do tempo

Description:

A função calcula o melhor caminho de um ponto A até B, com possibilidade de
paradas, sendo que os pontos alteram sua posição com o tempo. Na existência
de paradas, a rota será calculada de trecho em trecho, até o ponto de
destino. Será retornado um resumo do melhor caminho escolhido e um mapa
representando o trajeto.

Usage:

rota(posx, posy, t.partida=1, vel=5, p.partida, p.destino, p.parada=NULL,
```

```
ordem.fixa=TRUE, encurtar="tempo.espera", tx.amostragem=1, hora1=0, dia1=0,
xlab="Unidade de medida", ylab="Unidade de medida", main="Mapa do espaço")

Arguments:


posx
        data frame da posição dos pontos no eixo x, em cada horário
amostrado, com taxa de amostragem constante. A primeira coluna deve conter
os nomes dos pontos.

posy
        data frame da posição dos pontos no eixo y, em cada horário
amostrado, com taxa de amostragem constante. A primeira coluna deve conter
os nomes dos pontos.

t.partida
         número da coluna, nos data.frames posx e posy, correspondente ao
horário de partida. Descontar a primeira coluna (dos caracteres). Por
padrão, t.partida será igual a 1, ou seja, a primeira coluna de dados
numéricos.

vel
        velocidade do deslocamento,no formato: unidade de x e y/taxa de
amostragem

p.partida
        nome do ponto de partida (character).

p.destino
        nome do ponto de destino (character).

p.parada
        vetor do(s) nome(s) do(s) ponto(s) de parada, se existirem
(character).

ordem.fixa
        lógico; se FALSE, a ordem dos pontos de parada é calculada de forma
a otimizar o tempo ou a distância do deslocamento, a depender do critério de
"encurtar"; se TRUE, será utilizada a ordem dos pontos definida em p.parada.

encurtar
        critério pelo qual o caminho será escolhido ("tempo.espera" ou
"distancia"). Se encurtar = "tempo.espera", o caminho retornado será aquele
com menor duração e menor tempo de espera. Se encurtar = "distancia", o
caminho retornado será o mais curto, independente da duração do
deslocamento.

tx.amostragem
        frequência com que a posição dos pontos no espaço foi registrada.
```

```
Por padrão, tx.amostragem=1, o que significa que os registros foram feitos a
cada hora.

hora1
        horário de partida real. Por padrão, o horário de partida é 0 horas.
Os minutos e segundos devem ser representados como décimos. Por exemplo:
escrever 1,5 para se referir a 01:30.

dia1
        dia de partida real. Por padrão, o dia de partida é 0.

xlab
        legenda para o eixo x do mapa

ylab
        legenda para o eixo y do mapa

main
        título do mapa


Details:

O formato dos data.frames deve ser:

>posx
  point.name positionx.time1 positionx.time2 positionx.time3    ...
1  character        numeric        numeric        numeric
2  character        numeric        numeric        numeric
3  character        numeric        numeric        numeric
...

>posy
  point.name positiony.time1 positiony.time2 positiony.time3    ...
1  character        numeric        numeric        numeric
2  character        numeric        numeric        numeric
3  character        numeric        numeric        numeric
...

A primeira coluna, em formato "character", deve conter o nome dos pontos. As
colunas seguintes devem conter a posição de cada ponto no eixo x (posx) ou
eixo y (posy), em cada horário amostrado (time1, time2, time3, ...).
O tempo decorrido entre time1 e time2 equivale à tx.amostragem, que deve ser
constante. Por padrão, o horário de partida será 0, utilizando a primeira
coluna de amostras (t.partida=1). O valor 1 se refere à primeira coluna
numérica dos dois data.frames (positionx.time1 e positiony.time1).

Value:

Retorna uma lista contendo um data.frame com o resumo da melhor rota
encontrada, com base no critério escolhido em "encurtar". No caso de
```

```
empates, mais data.frames serão retornados, como outros itens na lista. Cada
data.frame conterá:

Nome.do.ponto   : Nome dos pontos presentes na rota, ordenados desde a
partida até o destino.

Posição.em.x    : Posição no eixo x de cada ponto, no horário de chegada em
cada um.

Posição.em.y    : Posição no eixo y de cada ponto, no horário de chegada em
cada um.

Horario.de.chegada  : Horário em que se chegará em cada ponto, baseado no
valor de hora1 e taxa de amostragem.

Tempo.de.espera : Intervalo de tempo entre a chegada do sujeito na posição,
calculada de acordo com a velocidade (vel), e a chegada do ponto nessa
posição.

Dia.de.chegada  : Dia em que se chegará em cada ponto, baseado no valor de
hora1, dia1 e taxa de amostragem

É retornado, também, um mapa ilustrando a rota desde a partida até o
destino, exibindo, para cada ponto, seu nome e horário de chegada. No caso
de empate, mais de um mapa será exibido.

Warning:

É importante que a velocidade do deslocamento seja suficiente para compensar
o movimento dos pontos. Se a velocidade for pequena (por exemplo, igual a
uma unidade de x e y por taxa de amostragem), é necessário que os pontos se
desloquem dentro de um espaço delimitado e que haja registro das posições
durante um tempo mais longo. Caso contrário, não haverá rotas para serem
retornadas.

Note:

O critério encurtar=distancia escolherá as rotas com as menores distâncias,
independente da duração da viagem. Portanto, o seu uso para data.frames com
registros de vários dias poderá aumentar o tempo de viagem em mais de 24
horas, dependendo das distâncias encontradas.

Author(s):

Milene Gomes Jannetti

Examples:

#### Criando dados ####
##Posicao x
```

```
set.seed(23)
posx<-data.frame(nome=letters, mresp=sample(seq(from=1, to=50, by=0.1),
(length(letters))), stringsAsFactors=FALSE)
for (i in 3:97)
{
    for (j in 1:(length(letters)))
    {
    posx[j,i]<-sample(round(rnorm(20,posx[j,2],5),1),1)
    }
}

##Posicao y
set.seed(22)
posy<-data.frame(nome=letters, mresp=sample(seq(from=1, to=50, by=0.1),
(length(letters))), stringsAsFactors=FALSE)
for (i in 3:97)
{
    for (j in 1:(length(letters)))
    {
    posy[j,i]<-sample(round(rnorm(20,posy[j,2],5),1),1)
    }
}

####Criando rota do ponto C até o ponto F, parando em A, L e G. ###

##Partida determinada para ocorrer em time3 (terceira coluna numérica dos
data.frames).
##A coluna time3 foi amostrada no horário 12:00 do dia 2, então hora1=12 e
dia1=2
##Velocidade do deslocamento é igual a 2 unidades de x e y/taxa de
amostragem.
##Taxa de amostragem é igual a 0.5, ou seja, 30 minutos.
##Deixar a função escolher a ordem dos pontos de parada (ordem.fixa=FALSE),
para otimizar o tempo do deslocamento (encurtar="tempo.espera").

rota(posx, posy, t.partida=3, vel=2, p.partida="c", p.destino="f",
p.parada=c("a","l","g"), tx.amostragem=0.5, hora1=12, dia1=2,
ordem.fixa=FALSE, encurtar="tempo.espera")
```