

Função Vizinhos

```
vizinhos <- function(locais,vizi,raio,min.vizi=0,croqui=TRUE)
  ##gera a função "vizinhos", com os argumentos "locais", "vizi", "raio",
  "min.vizi" e "croqui". Esta função foi desenvolvida no âmbito da disciplina
  de R/USP-2019
  { #abre os comandos da função
    ##iniciam-se os testes de premissa
    if(ncol(locais)!=2) #estabelece teste de premissa 1, de que o objeto
    "locais" deve possuir exatamente duas colunas
    { #apresenta consequência do argumento "locais" não possuir duas
    colunas
      stop("número de colunas do objeto 'locais' é diferente de
      2") #para de rodar a função e apresenta aviso
    } # fecha o teste de premissa 1
    if(class(locais[,1])!="numeric" | class(locais[,2])!="numeric")
    #estabelece teste de premissa 2, de que as duas colunas do objeto "locais"
    devem ser da classe "numeric"
    { #apresenta consequência de ao menos uma das colunas de
    "locais" não ser numérica
      stop("ao menos uma das colunas do objeto 'locais' não é
      numerica") #para de rodar a função e apresenta aviso
    } #fecha o teste de premissa 2
    if(ncol(vizi)!=2) #estabelece teste de premissa 3, de que o objeto
    "vizi" deve possuir exatamente duas colunas
    { #apresenta consequência do argumento "vizi" não possuir duas
    colunas
      stop("número de colunas do objeto 'vizi' é diferente de 2")
    #para de rodar a função e apresenta aviso
    } #fecha o teste de premissa 3
    if(class(vizi[,1])!="numeric" | class(vizi[,2])!="numeric")
    #estabelece teste de premissa 4, de que as duas colunas do objeto "vizi"
    devem ser da classe "numeric"
    { #apresenta consequência de ao menos uma das colunas de "vizi"
    não ser numérica
      stop("ao menos uma das colunas do objeto 'vizi' não é
      numerica") #para de rodar a função e apresenta aviso
    } #fecha o teste de premissa 4
    if(raio<=0) #estabelece teste de premissa 5, de que o argumento
    "raio" deve ter valor superior a zero
    { #apresenta consequência de o raio ser igual ou inferior a zero
      stop("o 'raio' informado deve ser maior que zero") #para de
      rodar a função e apresenta aviso
    } #fecha o teste de premissa 5
    ##iniciam-se as correções dos argumentos:
    if (sum(is.na(locais))>0) #controle de fluxo para verificar se há
    NAs no argumento "locais"
    { #abre o controle de fluxo para a consequência de haver NAs em
    "locais"
```

```
    locais <- na.omit(locais) #remove as linhas do argumento
"locais" que apresentarem NA para uma das duas coordenadas, e sobrescreve
"locais"

    cat("\n\t aviso: as linhas de 'locais' contendo NA(s) foram
excluídas \n\n") #apresenta aviso ao usuário
    } #fecha o controle de fluxo para quando há NAs em "locais"
    if (sum(is.na(vizi))>0) #controle de fluxo para verificar se há NAs
no argumento "vizi"
    { #abre o controle de fluxo para a consequência de haver NAs em
"vizi"

        vizi <- na.omit(vizi) #remove as linhas do argumento "vizi" que
apresentarem NA para uma das duas coordenadas, e sobrescreve "vizi"
        cat("\n\t aviso: as linhas de 'vizi' contendo NA(s) foram
excluídas \n\n") #apresenta aviso ao usuário
        } #fecha o controle de fluxo para quando há NAs em "vizi"
        ##iniciam-se os cálculos a partir dos argumentos informados
        n.locais <- nrow(locais) #cria objeto contendo o número de locais
informado, extraído do argumento "locais"
        n.vizi <- nrow(vizi) #cria objeto contendo o número de vizinhos
informado, extraído do argumento "vizi"
        dists <- matrix(NA, ncol=n.vizi, nrow=n.locais) #gera a matriz de
distâncias, que posteriormente receberá os valores das distâncias entre cada
local (linhas da matriz) e cada um dos vizinhos (colunas da matriz). Assim,
o número de linhas desta matriz corresponderá ao número de locais, enquanto
o número de colunas corresponde ao número de vizinhos


for(i in 1:n.vizi) #gera o primeiro ciclo for para calcular as distâncias
entre todos os "i" vizinhos e cada local informado
{ #abre o primeiro ciclo "for"
    for(j in 1:n.locais) #gera o segundo ciclo for, dentro do primeiro,
para repetir o cálculo de distância entre os "j" locais aqui indicados e
todos os "i" vizinhos indicados no primeiro ciclo
        { #abre o segundo ciclo "for". Os três próximos comandos são o
cálculo da distância euclidiana entre dois pontos.
            dif.x2=(locais[j,1]-vizi[i,1])^2 #calcula o quadrado da
diferença entre os valores de x do local e do vizinho testados
            dif.y2=(locais[j,2]-vizi[i,2])^2 #calcula o quadrado da
diferença entre os valores de y do local e do vizinho testados
            dists[j,i]<-sqrt(dif.x2 + dif.y2) #calcula a raiz da soma
dos quadrados dos catetos, chegando à distância euclidiana entre dois
pontos, sendo que um dos pontos corresponde a um local testado e o outro
corresponde a um vizinho testado
        } #fecha o segundo ciclo "for"
    } #fecha o primeiro ciclo "for"
    rank <- locais #cria novo data frame "rank" a partir do objeto
"locais"

    rank$n.vizi <- rep(NA,n.locais) #cria nova coluna no data frame
"rank", inicialmente preenchida por "NA"s
    for (i in 1:n.locais) #gera ciclo com "i" elementos, para realizar
repetições em número igual ao número de locais. O objetivo é preencher, em
```

```
cada ciclo, uma das linhas da nova coluna "n.vizi".
  { #abre o ciclo "for"
    rank$n.vizi[i] <- sum(dists[i,]<=raio) #preenche a nova coluna
    "n.vizi" com o número de vizinhos de cada local que estão incluídos dentro
    do raio informado. O cálculo é feito a partir da soma do seguinte teste
    lógico: quais distâncias calculada na matriz "dists" são menores ou iguais
    ao valor informado no argumento "raio"?
  } #fecha o ciclo "for"
  rank <- rank[order(-rank[,3]), ] #ranqueia os locais em ordem
  decrescente com relação ao número de vizinhos englobados no "raio"
  informado. A terceira coluna do data frame "rank" é "n.vizi", e o símbolo "-"
  indica ordem decrescente. O data frame reordenado sobrescreve o objeto
  "rank" anterior.
  if(min.vizi) #controle de fluxo para especificar os casos em que
  tiver sido informado valor (deve ser inteiro) para o argumento "min.vizi"
  { #abre o controle de fluxo para quando "min.vizi" foi informado
    if(class(min.vizi)!="integer") #cria controle de fluxo para
    verificar se o argumento "min.vizi" não é da classe "integer"
    { #apresenta consequência de resultado de "min.vizi" não ser da
    classe "integer"
      min.vizi <- as.integer(min.vizi) #converte o argumento
      "min.vizi" para a classe "integer"
      cat("\n\t aviso: o argumento 'min.vizi' foi convertido para
      a classe 'integer', resultando no número inteiro=",min.vizi,"\n\n")
      #apresenta aviso ao usuário
    } #fecha o controle de fluxo da conversão de "min.vizi" em
    "integer"
    if(min.vizi<0) #cria o teste de premissa que exige que o
    argumento "min.vizi" seja maior que zero
    { #apresenta consequência de "min.vizi" ser negativo
      stop("o argumento 'min.vizi' deve ser maior que zero") #para
      de rodar a função e apresenta aviso
    } #fecha o teste de premissa de que "min.vizi" deve ser positivo
    rank <- rank[rank[,3] >= min.vizi, ] #sobrescreve o data frame
    "rank", apresentando apenas as linhas com valor de "n.vizi" maior ou igual
    ao argumento "min.vizi".
    locais <- rank[, -3] # sobrescreve o data frame "locais",
    apresentando apenas os locais com valor de "n.vizi" maior ou igual ao
    argumento "min.vizi", mas sem a coluna "n.vizi" que foi gerada para o objeto
    "rank". O objeto "locais" sobrescrito será usado posteriormente para gerar o
    croqui.
    n.locais <- nrow(locais) #sobrescreve o objeto "n.locais", para o
    atualizar quanto ao número de locais já ranqueados.
  } #fecha o controle de fluxo para quando "min.vizi" foi informado
  if(croqui) #controle de fluxo para gerar o croqui a critério do
  usuário. O padrão é croqui=TRUE.
  { #abre o controle de fluxo para gerar o croqui
    if(class(croqui)!="logical") # estabelece teste de premissa de
    que o argumento "croqui" deve ser da classe "logical"
    { #apresenta consequência de o argumento "croqui" não ser da
    classe "logical"
```

```
stop("o argumento 'croqui' deve ser da classe 'logical'")
#para de rodar a função e apresenta aviso
} # fecha o teste de premissa de "croqui" ser "logical"
##a partir daqui iniciam-se os comandos para visualização
gráfica do croqui
plot(vizi, col="green", pch=20, ylab="y", xlab="x") #plota as
coordenadas x e y do data frame "locais", já considerando que as coordenadas
x estão na primeira coluna e que as coordenadas y estão na segunda coluna.
Os vizinhos aparecerão como círculos pequenos preenchidos na cor verde. Os
nomes dos eixos constarão simplesmente como "x" e "y".
points(locais,col="blue",pch=15) #plota as coordenadas x e y do
data frame "locais", já considerando que as coordenadas x estão na primeira
coluna e que as coordenadas y estão na segunda coluna. Os locais aparecerão
como quadrados preenchidos na cor azul.
require(plotrix) #chama o pacote "plotrix", que contém a função
"draw.circle" usada no "for" a seguir
for(i in 1:n.locais) #cria um ciclo "for" com o número de
repetições equivalente ao número de locais ranqueados
{ #abre o ciclo de repetições para gerar as circunferências em
torno dos locais ranqueados.
draw.circle(x=locais[i,1],y=locais[i,2],radius=raio,nv=10000,border=NULL,col
=rgb(1,0,0,0.3),lty=1) #desenha circunferências com o raio informado no
argumento "raio" em torno dos locais ranqueados. Como parâmetros gráficos
para o desenho, as circunferências terão 10000 vértices, linha contínua
preta definindo o perímetro, e estarão preenchidas pela cor vermelha, com
transparência de 30%.
} #fecha o ciclo "for" para gerar as circunferências em torno
dos locais ranqueados.
if(require(plotrix)!=TRUE) #estabelece o teste de premissa de
que o pacote 'plotrix' deve estar instalado
{ #apresenta consequência do pacote 'plotrix' não estar
instalado
stop("o argumento 'croqui' requer a instalação do pacote
'plotrix'") #para a função e apresenta aviso
} #fecha o teste de premissa da instalação do pacote 'plotrix'
} #fecha o controle de fluxo para gerar o croqui
colnames(rank) <- c("coord.x","coord.y","número de vizinhos")
#altera nomes das colunas de "rank" para facilitar a compreensão do
resultado
cat("\n Locais ranqueados quanto ao número de vizinhos: \n\n")
#insere texto informando do que se trata a tabela gerada como produto da
função
return(rank) #retorna, como principal produto da função, o objeto
"rank", que consiste na tabela ranqueando os locais (pontos) em ordem
decrescente com relação ao número de vizinhos englobados no "raio" de
distância de cada local, já excluindo os locais com número de vizinho
inferior a "min.vizi"
} #fecha as linhas de comando da função "vizinhos"
```

From:
<http://ecor.ib.usp.br/> - **ecoR**

Permanent link:
http://ecor.ib.usp.br/doku.php?id=05_curso_antigo:r2019:alunos:trabalho_final:rafael.chaves:funcao_vizinhos 

Last update: **2020/08/12 06:04**