Proposta de função final

Plano B: Calculo de L de Ripley

OBS: selecionar a funçao toda para rodar.

No rstudio, ao executar apenas a primeira linha (function()), estava bugando.

```
pad.esp = function(x, y, lim, rmax = max((lim[2]-lim[1]),(lim[4]-lim[3]))/2,
r = 1, ic = TRUE, conf = c(0.05, 0.95), nsim = 100, plot = TRUE) #função com
os argumentos x, y, lim, rmax, r, ic, conf e nsim, sendo o defaut da função
especificado por "= ..."
{
  #######Verificação de parametros#########
  if(sum(x<0) !=0 |class(x)!="numeric"&class(x)!= "integer") #se x for menor
que zero ou a classe for diferente de numeric e integer,
  {stop("x deve ser numerico ou inteiro e maior que 0.")} #para e imprime a
mensagem
  if(sum(y<0) !=0 |class(y)!="numeric"&class(y)!= "integer") #se y for menor
que zero ou a classe for diferente de numeric e integer,
  {stop("y deve ser numerico ou inteiro e maior que 0.")} #para e imprime a
mensagem
  if(length(x)!=length(y)) #se o comprimento de x e y é diferente
  {stop("x e y devem ter o mesmo comprimento")} #para e imprime a mensagem
  if(length(lim)!=4|class(lim)!="numeric"&class(lim)!= "integer") #se o
comprimento de lim não é 4 e a classe não é numeric e integer,
  {stop("lim deve ser numerico ou inteiro e com comprimento 4")} #para e
imprime a mensagem
  if(length(rmax)!=1|class(rmax)!= "integer"&class(rmax)!="numeric") #se o
comprimento de lim não é 1 e a classe não é numeric e integer,
  {stop("rmax deve ser inteiro ou numerico e de comprimento 1")} #para e
imprime a mensagem
  if(length(r)!=1|class(r)!= "integer"&class(r)!="numeric") #se o
comprimento de r não é 1 e a classe não é numeric e integer,
  {stop("r deve ser inteiro e de comprimento 1")} #para e imprime a mensagem
  if(length(nsim)!=1|class(nsim)!= "integer") #se o comprimento de nsim não
é 4 e a classe não é integer,
  {warning("nsim deve ser inteiro e de comprimento 1")} #imprime a mensagem
de aviso
  if(class(ic)!="logical") #se ic não é logico
  {stop("ic deve ser logico")} #para e imprime a mensagem
  if(class(plot)!="logical") #se plot não é logico
  {stop("plot deve ser logico")} #para e imprime a mensagem
  if(r>=rmax) #se r é maior ou iguala rmax
  {stop("rmax deve ser maior que r")} #para e imprime a mensagem
  if(conf[1]>conf[2]) #se a segunda posição de ic for maior que a primeira
```

```
  {stop("ic deve conter os quantis minimo e máximo, respectivamente")} #para
e imprime a mensagem
  ########## Criando objetos prévios ###########
  distancia = matrix(NA, ncol = length(x), nrow = length(y)) #cria matriz
que irá guardar a distancia de cada ponto, com número de linhas e colunas
igual ao comprimento de x e y.
  ntot = length(x) #guarda comprimento de x
  dens = ntot/((lim[2] - lim[1])*(lim[4] - lim[3])) #calcula a densidade de
pontos na area de estudo.
  sequ = seq(from = 1, to = rmax, by = r) #guarda o valor dos raios a serem
utilizados em oring
  ind = matrix(rep(NA, times = rmax*ntot), nrow = length(1:rmax), ncol =
ntot) #cria matriz que irá guardar quantos pontos são menores que um
determinado raio, com NAs. O numero de linhas é rmax e o de colunas ntot
  dimnames(ind) = list(paste("r", 1:rmax, sep = ""), paste("p", 1:ntot, sep
= "")) #altera em ind os nomes das linhas para o raio (r) e colunas para o
ponto (p) correspondente.
  ############Calculos para os meus dados###########
  distx = as.matrix(dist(x, upper=T)) #cria matriz de distancias em x
  disty = as.matrix(dist(y, upper=T)) #cria matriz de distancias em y
  distancia = sqrt((distx^2 + disty^2)) #calcula a distância euclidiana
entre todos os pontos
  for(i in 1:rmax) #ciclo for de 1 a rmax
  {
    ind[i,] = apply(distancia, 2, FUN = function(x)
{sum(x<c(1:rmax)[i]&x!=0)}) #aplica a função que conta quantos objetos são
menores que um determinado raio a cada linha de distancia
  }
  kr = apply(ind, 1, sum)*as.numeric((1/(ntot*dens))) #calculo para o k de
ripley: soma todos os pontos para cada raio e multiplica por
((1/(ntot*dens)))
  rip = sqrt(kr/pi) - 1:rmax #calculo para l de ripley
  tempring = rip[sequ] #extrai as posições sequ de ring
  ring = diff(tempring) #subtrai um raio do anterior
  ##########calculo do IC ############
  if(ic==TRUE) #se ic = TRUE
  {
    indsim = matrix(rep(NA, times = (rmax*ntot)), nrow = length(1:rmax),
ncol = ntot) #cria matriz que irá guardar quantos pontos são menores que um
determinado raio, com NAs. O numero de linhas é rmax e o de colunas ntot.
vai ser recriado a cada simulação
    dimnames(indsim) = list(paste("r", 1:rmax, sep = ""), paste("p", 1:ntot,
sep = "")) #altera em ind os nomes das linhas para o raio (r) e colunas para
o ponto (p) correspondente.
    ripsim = matrix(NA, ncol = rmax, nrow = nsim) # cria matriz contendo
ntot colunas, nsim linhas, contendo NAs. Vai guardar os valores de l de
ripley a cada simulação
    dimnames(ripsim) = list(paste("s", 1:nsim, sep = ""), paste("r", 1:rmax,
sep = "")) #altera em indsim os nomes das linhas para a simulação (s) e
colunas para o raio (r) correspondente.
```

```
    tempringsim = ripsim[, sequ] #guarda as colunas sequ de ripsim
    ringsim = matrix(NA, ncol = length(sequ)-1, nrow = nsim) # cria matriz
contendo comprimento de sequ colunas, nsim linhas, contendo NAs. Vai guardar
os valores de oring a cada simulação.
    dimnames(ringsim) = list(paste("s", 1:nsim, sep = ""), paste("r",
sequ[-1], sep = "")) #altera em indsim os nomes das linhas para a simulção
(s) e colunas para o raio (r) correspondente.
    ripsim[1,] = rip #guarda rip na primeira linha de ripsim
    ringsim[1,] = ring #guarda ring na primeira linha de ringsim
    for(k in 2:nsim) #ciclo for de 2 a nsim
    {
      xsim = runif(ntot, lim[1], lim[2]) #sorteia números de uma
distribuição aleatoria entre os limites de x
      ysim = runif(ntot, lim[3], lim[4]) #sorteia números de uma
distribuição aleatoria entre os limites de y
      distxsim = as.matrix(dist(xsim, upper = T)) #matriz de distâncias em x
      distysim = as.matrix(dist(ysim, upper = T)) #matriz de distâncias em y
      distsim = sqrt((distxsim^2 + distysim^2)) #calculo da distancia
euclidiana entre todos os pontos
      for(l in 1:rmax) #ciclo for de 1 a rmax
      {
        indsim[l,] = apply(distsim, 2, FUN = function(x)
{sum(x<c(1:rmax)[l]&x!=0)}) #aplica a função que conta quantos objetos são
menores que um determinado raio a cada linha de distancia
        krsim = apply(indsim, 1, sum)*as.numeric(1/(ntot*dens)) #calculo
para o k de ripley: soma todos os pontos para cada raio e multiplica por
(1/(ntot*dens)
        ripsim[k,] = sqrt(krsim/pi) - 1:rmax #calculo para l de ripley
        tempringsim[k,] = ripsim[k,sequ] #guarda colunas sequ de ripsim
        ringsim[k,] = diff(tempringsim[k,]) #subtrai uma coluna pela
anterior
      }
    }
    ##########comparando dados com envelope de confiança###############
    ripquantis = matrix(NA, ncol = 2, nrow = rmax) #cria matriz com NAs de 2
colunas e rmax linhas
    dimnames(ripquantis) = list(paste("r", 1:rmax, sep = ""), conf) #altera
o nome das dimensões para o raio e conf
    ringquantis = matrix(NA, ncol = 2, nrow = length(sequ)-1) #cria matriz
com NAs de 2 colunas e comprimento de sequ-1 linhas
    dimnames(ringquantis) = list(paste("r", sequ[-1], sep = ""), conf)
#altera o nome das dimensões para o raio e conf
    for(m in 1:rmax) #ciclo for de 1 a rmax
    {ripquantis[m,] = quantile(ripsim[,m], probs = conf)} #calcula quantis
conf para cada raio
    for(n in 1:length(sequ)-1) #ciclo for de 1 a comprimento de sequ-1
    {ringquantis[n,] = quantile(ringsim[,n], probs = conf)}  #calcula
quantis conf para cada raio
    rippad = rip #cria objeto rippad igual a rip
    rippad[which(rip<ripquantis[,1])] = "uniforme" #atribui "uniforme" para
todos os rippad que são menores que ripquantis (limite inferior do
```

```
intervalo)
    rippad[which(rip>ripquantis[,2])] = "agregado" #atribui "agregado" para
todos os rippad que são maiores que ripquantis (limite superior do
intervalo)
    rippad[which(rip>=ripquantis[,1]&rip<=ripquantis[,2])] = "aleatório"
#atribui "aleatorio" para todos os rippad que estão entre o IC
    ringpad = ring #cria objeto ringpad igual a rip
    ringpad[which(ring<ringquantis[,1])] = "uniforme" #atribui "uniforme"
para todos os ringpad que são menores que ringquantis (limite inferior do
intervalo)
    ringpad[which(ring>ringquantis[,2])] = "agregado"  #atribui "agregado"
para todos os ringpad que são maiores que ringquantis (limite superior do
intervalo)
    ringpad[which(ring>=ringquantis[,1]&ring<=ringquantis[,2])] =
"aleatório" #atribui "aleatorio" para todos os ringpad que estão entre os
limites de IC
    valores = list(ripley = rip, IC_ripley = ripquantis, oring = ring,
IC_oring = ringquantis) #cria lista com os valores calculados e simulados.
    ##############analise gráfica##############
    if(plot==TRUE) #se plot = TRUE
    {
      par(mfrow = c(1,2)) #divide a janela grafica em 2
      plot(c(1:rmax), rip, pch = 20, col = "red", xlab = "Raio", ylab = "L
de Ripley") #plota rip~rmax, alterando tipo de ponto e cor
      lines(1:rmax, rip, lwd = 2, col = "red") #adiciona linha ligando os
pontos plotados anteriormente
      lines(1:rmax,ripquantis[,1], lty = 2, lwd = 2) #adiciona linha do IC
inferior
      lines(1:rmax, ripquantis[,2], lty = 2, lwd = 2) #adiciona linha do IC
superior
      abline(0, 0, lwd = 2) #adiciona linha horizontal em 0, para comparação
      legend(rmax-10, max(rip)-diff(range(rip))*0.02, legend = c("L.
Ripley", "Env. Conf.", "zero"), bty = "n", lty = c(1, 2, 1), col = c("red",
"black", "black"), cex = 0.7, x.intersp = 0.2, lwd = 2, seg.len = 1)
#adiciona legenda para as linhas, mudando tipo de linha, cor, tamanho,
espaço entre a lnha e o texto, largura da linha, comprimento do segmento
      plot(sequ[-1], ring, pch = 20, col = "red", xlab = "Raio", ylab = "O-
Ring") #plota ring~rmax, alterando tipo de ponto e cor
      lines(sequ[-1], ring, lwd = 2, col = "red") #adiciona linha ligando os
pontos plotados anteriormente
      lines(sequ[-1],ringquantis[,1], lty = 2, lwd = 2) #adiciona linha do
IC inferior
      lines(sequ[-1], ringquantis[,2], lty = 2, lwd = 2) #adiciona linha do
IC superior
      lines(sequ[-1], rep(dens, times = length(sequ)-1), lwd = 2) #adiciona
linha horizontal em dens, para comparação
      text(rmax-3, dens+0.1, expression(lambda)) #adciona texto "lambda" em
cima da linha horizontal em dens
      legend(rmax-10, max(ring)-diff(range(ring))*0.02, legend = c("O-ring",
"Env. Conf.", expression(lambda)), bty = "n", lty = c(1, 2, 1), lwd = 2, col
```

```
= c("red", "black", "black"), cex = 0.7, x.intersp = 0.2, seg.len = 1)
#adiciona legenda para as linhas, mudando tipo de linha, cor, tamanho,
espaço entre a lnha e o texto, largura da linha, comprimento do segmento
      par(mfrow = c(1,1)) #divide a janela grafica em 1}
    }
    return(list(L_de_Ripley = rippad, O_Ring = ringpad, Valores = valores))
#retorna lista com rippad, ringpad e valores, alterando o nome dos elementos
da lista.
  }
  else{ #se ic = false
    if(plot==TRUE) #se plot = TRUE
    {
      par(mfrow = c(1,2)) #divide a janela grafica em 2
      plot(1:rmax, rip, pch = 20, col = "red", xlab = "Raio", ylab = "L de
Ripley") #plota rip~rmax, alterando tipo de ponto e cor
      lines(1:rmax, rip, lwd = 2, col = "red") #adiciona linha ligando os
pontos plotados anteriormente
      abline(0, 0, lwd = 2) #adiciona linha horizontal em 0, para comparação
      legend(rmax-10, max(rip)-diff(range(rip))*0.02, legend = c("L.
Ripley", "zero"), bty = "n", lty = c(1, 1), col = c("red", "black"), cex =
0.7, x.intersp = 0.2, lwd = 2, seg.len = 1) #adiciona legenda para as
linhas, mudando tipo de linha, cor, tamanho, espaço entre a lnha e o texto,
largura da linha, comprimento do segmento
      plot(sequ[-1], ring, pch = 20, col = "red", xlab = "Raio", ylab = "O-
Ring") #plota ring~rmax, alterando tipo de ponto e cor
      lines(sequ[-1], ring, lwd = 2, col = "red") #adiciona linha ligando os
pontos plotados anteriormente
      lines(sequ[-1], rep(dens, times = length(sequ)-1), lwd = 2) #adiciona
linha horizontal em dens, para comparação
      text(rmax-3, dens+0.1, expression(lambda)) #adciona texto "lambda" em
cima da linha horizontal em dens
      legend(rmax-10, max(ring)-diff(range(ring))*0.02, legend = c("O-ring",
expression(lambda)), bty = "n", lty = c(1, 1), lwd = 2, col = c("red",
"black"), cex = 0.7, x.intersp = 0.2, seg.len = 1)
      par(mfrow = c(1,1)) #divide a janela grafica em 1
    }
  }
  return(list(L_de_Ripley = rip, O_Ring = ring)) #retorna lista com rip e
ring mudando o nome dos elementos da lista.
}
```