Instruções para utilização do algoritmo para cálculo da área das parcelas RAPELD em curva de nível

Elaboração do script: David Dias
Elaboração das instruções: Adriane Morais **(Modificado por Sergio Santorelli Junior em 20/01/2014; E-mail: santorelli.jr@gmail.com)**

Esse script foi elaborado e disponibilizado com intuito de facilitar o cálculo da área das parcelas por pesquisadores e alunos que utilizam as parcelas RAPELD do Programa de Pesquisa em Biodiversidade – PPBio. Para que a função execute corretamente o cálculo, alguns cuidados devem ser tomados.

Entrada dos dados

A tabela de entrada dos dados devem seguir o seguinte formato:

Exemplo:

	plot_id	azimute	comp.seg	remove
1	1	0.000000	10	0
2	1	0.000000	10	0
3	1	0.000000	10	0
4	1	0.000000	10	0
5	1	0.000000	10	0
6	1	0.000000	10	0

I. A coluna ***plot_id*** refere-se a identificação da parcela;
II. Na coluna ***azimute*** deverá conter os azimutes coletados em cada segmento;
III. ***comp.seg*** indicará na tabela o comprimento de cada segmento; e
IV. A coluna ***remove*** é utilizada para identificar os segmentos da parcela que são desconsiderados (removidos) por algum motivo, por exemplo, "*segmento cortando a trilha*". Deve ser preenchida com "0" se o trecho não foi removido da linha central e "1" se o trecho foi removido.

Entendendo o script

Para realizar o cálculo da área das parcelas é utilizada a função chamada **doArea( ).** Mas antes, uma série de outras funções complementares que auxiliarão nos cálculos deverão ser executadas. Você

não deve se preocupar em aplicar essas funções com seus dados, elas serão automaticamente aplicadas dentro da função **doArea( )**.

A primeira função apresentada no script é chamada **doCentraline( )**, essa função é responsável por criar a linha central da parcela. A saída da função é uma lista contendo as coordenadas para plotagem e os ângulos em radianos (nesse formato que é reconhecido no software R).

A segunda função é chamada **ang( )**, essa função calcula o ângulo utilizando a lei dos cossenos. Essa função será utilizada para calcular a área posteriormente. Os cálculos desses ângulos são feito em função da parcela e não somente da linha central.

A terceira função tem o nome **findd( )**, essa função verifica se existe cruzamento em dois segmentos de linha. Essa função confere se os segmentos de linha da parcela foram plotados na mesma direção que os segmentos da linha central (De tal maneira que estas linhas não se cruzem). Referência: http://local.wasp.uwa.edu.au/~pbourke/geometry/lineline2d/

A quarta função; **Support( )** é utilizada apenas para auxiliar no cálculo das áreas.

A quinta função informa se o polígono é côncavo ou convexo. É necessário verificar se o polígono é côncavo ou convexo pois é feito o cálculo da área do semicírculo e o programa precisa reconhecer onde estão os semicírculos. O nome dessa função é **concave( )**.

A função **calcArea( )** também é uma função auxiliar utilizada para calcular a área das parcelas.

A sétima função do script [**plotParcela( )**] cria o gráfico da parcela parâmetros semelhantes a calcArea incluindo o resultado de **calcArea** no parâmetro <area>.

Por fim, a função **doArea( ).** Essa função deverá ser executada após a execução de todas as outras funções acima comentadas.

Utilizando o script

Os comandos abaixo criarão um conjunto de dados para servir de exemplo.

```
dados = data.frame(plot_id = rep(c(1, 2), each = 10), azimute=c(rep(0, 10), runif(10, 0, 100)), segmento = rep(10, 10), remove = rep(0, 10))
dados[sample(12:19, 1),'remove'] = 1
```

Exemplo de uma tabela criada com o comando acima (sua tabela no R deverá ser semelhante a essa):

	plot_id	azimute	segmento	remove
1	1	0.000000	10	0
2	1	0.000000	10	0
3	1	0.000000	10	0
4	1	0.000000	10	0
5	1	0.000000	10	0
6	1	0.000000	10	0
7	1	0.000000	10	0
8	1	0.000000	10	0
9	1	0.000000	10	0
10	1	0.000000	10	0
11	2	10.793017	10	0
12	2	95.855447	10	0
13	2	22.537246	10	0
14	2	24.925612	10	0
15	2	19.686732	10	0
16	2	15.230459	10	0
17	2	9.586708	10	1

Para utilizar a função, copie e cole todo script abaixo no R e execute. Você perceberá no script todas a funções comentadas anteriormente (doCentraline( ), ang( ), concave( )...).

Início da função, execute a partir daqui (vá antes até o final do script e veja onde a função termina!...##

```
doCentraline <- function(data) {
  xys <- data.frame(x=0, y=0)
  thetas <- NULL
  for (i in 2:nrow(data)) {
    theta <- data$azimute[i - 1] * pi/180
    xys[i,] <- c(xys[i-1, 1] + data$segmento[i] * cos(theta),
                 xys[i-1, 2] + data$segmento[i] * sin(theta))
    thetas[i - 1] <- theta
  }
  list(coordenadas = xys, angulos = c(thetas, thetas[length(thetas)]), remove = data$remove)
}


# Função corretora de ângulos
# Se maior que 360 remova 360
# Se menor que 0 adicione 360
# Ex.: 400 - 360 = 40
#                                -5 + 360 = 355

correctAng <- function(x) {
  if (is.na(x)) {
    return(NA)
  } else if (x > (2 * pi)) {
    return (x - 2 * pi)
  } else if (x < 0) {
    return (x + 2 * pi)
  } else {
    x
  }
}


# Função que calcula o ângulo utilizando a lei dos cossenos

ang <- function(coord) {
  ang3 <- NULL
  if (all(coord[,2] == coord[1,2])) {
    if (coord[2,1] > coord[3,1]) {
      return(c(NA, pi))
    } else {
      return(c(NA, 0))
```

```
    }
  }
  for (i in 2:(nrow(coord) - 1)) {
    b <- dist(coord[c(i - 1, i),])
    cc <- dist(coord[c(i, i+1),])
    a <- dist(coord[c(i - 1, i + 1),])
    ang3[i] <- acos((b^2 + cc^2  - a^2) / (2 * b * cc))
    if (is.na(ang3[i])) {
      return(c(NA, pi))
    }
  }
  ang3
}

# Função que verifica se o ângulo é menor que <angle> e retorna um vetor
#                       contendo TRUE onde ocorre o ângulo menor que <angle>.

angLess <- function(coord, angle) {
  col <- rep(FALSE, nrow(coord))
  ang3 <- NULL
  for (i in 2:(nrow(coord) - 1)) {
    b <- dist(coord[c(i - 1, i),])
    cc <- dist(coord[c(i, i+1),])
    a <- dist(coord[c(i - 1, i + 1),])
    ang3[i] <- acos((b^2 + cc^2  - a^2) / (2 * b * cc))
    if (is.na(ang3[i])) {
      next
    }
    if (ang3[i] < (angle*pi/180)) {
      col[(i):(i+1)] <- TRUE
    }
  }
  col
}

# Função que verifica se existe cruzamentos em dois segmentos de linha
# referência http://local.wasp.uwa.edu.au/~pbourke/geometry/lineline2d/
findd <- function(a, b) {
  if (is.na(a) || is.na(b)) return(FALSE)
  deno <- (b[2,2] - b[1,2]) * (a[2,1] - a[1,1]) - (b[2,1] - b[1,1]) * (a[2,2] - a[1,2])
  if (deno == 0) return(FALSE)
  ua <- (  (b[2,1] - b[1,1]) * (a[1,2] - b[1,2]) - (b[2,2] - b[1,2]) * (a[1,1] - b[1,1]) ) / deno
  ub <- (  (a[2,1] - a[1,1]) * (a[1,2] - b[1,2]) - (a[2,2] - a[1,2]) * (a[1,1] - b[1,1]) ) / deno
  if (ua >= 0 && ua <= 1 && ub >= 0 && ub <= 1) return(TRUE)
  return(FALSE)
}

# Função auxiliar para o calculo das áreas. Não deve ser chamada manualmente, apenas por outra
função (doArea)
Support <- function(top, sut) {
  topSupportPoints <- data.frame(x = 0, y = 0)
```

```
  for (j in 1:(nrow(top)-1)) {
   if (j == 1 ) {
     c4 <- dist(rbind(top[j,],sut[j,],top[j+1,]))[-3]
     switch(
       which.min(c4),
{
  topSupportPoints = rbind(topSupportPoints, top[j,], sut[j,])
},
{
  topSupportPoints = rbind(topSupportPoints, top[j,], top[j+1,])
}
     )
   } else {
     c4 <- dist(rbind(sut[j-1,],top[j,],top[j+1,], sut[j,]))[-c(1,6)]
     switch(
       which.min(c4),
       # 1
{
  topSupportPoints = rbind(topSupportPoints, sut[j - 1, ], top[j + 1,])
},
       # 2
{
  topSupportPoints = rbind(topSupportPoints,sut[j - 1,], sut[j,])
},
       # 3
{
  topSupportPoints = rbind(topSupportPoints,top[j,], top[j + 1,])
},
       # 4
{
  topSupportPoints = rbind(topSupportPoints,top[j,], sut[j,])
}
     )
   }
  }
  topSupportPoints[-1,]
}

# Função que informa se o poligno é côncavo ou convexo
# Importante para saber se o ângulo formula uma entrada ou não no poligno
concave <- function(a, b, c) {
  # -1 = concave
  #  1 = convex
  #  0 = no side, colinear
  d = ((b[1] - a[1]) * (c[2] - a[2]) - (b[2] - a[2]) * (c[1] - a[1]))
  if ( d > 0 ) {
    return(1)
  } else if ( d < 0) {
    return(-1)
  }
  return (0)
```

```
}

# Calcula a área da parcela excluído os segmentos com ângulo menor que
#                    <lessAngle> e que <remove> for igual a 1 (um). O parametro <d> pode assumir 'top',
'bot' e 'both'.
#          'both': area de ambos os lados da parcela
# 'top': area do lado 'superior'
# 'bot': area do lado 'superior'
# 'sut': pontos de suporte para o lado norte fornecidos por doArea
# 'sub': pontos de suporte para o  lado norte fornecidos por doArea

calcArea <- function(trilha, top, bot, sut, sub, lessAngle, d = 'both') {
  conc = NULL
  for (i in 2:(nrow(trilha$coordenadas) - 1) ) {
    if (all(trilha$angulos[(i-1):(i)] == trilha$angulos[i] )) {
      conc[i] = 0
    } else {
      conc[i] = concave(trilha$coordenadas[i-1,],trilha$coordenadas[i,],trilha$coordenadas[i+1,])
    }
  }
  stop = Support(top, sut)
  sbot = Support(bot, sub)

  P <- angLess(trilha$coordenadas, lessAngle)
  area <- 0
  for (i in 1:(nrow(trilha$coordenadas) - 1)) {
    if (!all(P[i:(i+1)] == T) && trilha$remove[i+1] == 0 ) {
      if ( d == 'both') {
        return(calcArea(trilha, top, bot, sut, sub, lessAngle, d = 'top') +
               calcArea(trilha, top, bot, sut, sub, lessAngle, d = 'bot'))
      }
      if (d == 'top' ) {
        if ( i > 1 && i < (nrow(trilha$coordenadas) - 1) ) {
          if (conc[i] == -1 && conc[i+1] == -1) { # se o proximo e o atual são concavos
            tempr = rbind(trilha$coordenadas[i:(i+1),], stop[((i+1)*2-2):(i*2-1),], trilha$coordenadas[i,])
            ta <- sum(tempr[1:(nrow(tempr)-1), 1] * tempr[2:nrow(tempr), 2]) -
sum(tempr[1:(nrow(tempr)-1), 2] * tempr[2:nrow(tempr), 1])
            area = area + ta/2 + ( dist(rbind(trilha$coordenadas[i,],top[i,]))^2 * ang(rbind(stop[(i+1)*2-
2,],trilha$coordenadas[i+1,],stop[(i+1)*2-1,]))[2] )/2
          } else if (conc[i] == -1 && conc[i+1] > -1 ) { # se o proximo não é concavo mas o atual é
            tempr = rbind(trilha$coordenadas[i:(i+1),],top[i+1,],stop[i*2-1,],trilha$coordenadas[i,])
            ta <- sum(tempr[1:(nrow(tempr)-1), 1] * tempr[2:nrow(tempr), 2]) -
sum(tempr[1:(nrow(tempr)-1), 2] * tempr[2:nrow(tempr), 1])
            area = area + ta/2
          } else if (conc[i] > -1 && conc[i+1] == -1 ) { #se o proximo é concavo mas o atual não
            tempr = rbind(trilha$coordenadas[i:(i+1),],stop[(i+1)*2-2,], top[i,], trilha$coordenadas[i,])
            ta <- sum(tempr[1:(nrow(tempr)-1), 1] * tempr[2:nrow(tempr), 2]) -
sum(tempr[1:(nrow(tempr)-1), 2] * tempr[2:nrow(tempr), 1])
            area = area + ta/2 + ( dist(rbind(trilha$coordenadas[i,],top[i,]))^2 * ang(rbind(stop[(i+1)*2-
2,],trilha$coordenadas[i+1,],stop[(i+1)*2-1,]))[2] )/2
          } else { # se nenhum deles é concavo
```

```
        tempr = rbind(trilha$coordenadas[i:(i+1),], top[(i+1):i,], trilha$coordenadas[i,])
        ta <- sum(tempr[1:(nrow(tempr)-1), 1] * tempr[2:nrow(tempr), 2]) -
sum(tempr[1:(nrow(tempr)-1), 2] * tempr[2:nrow(tempr), 1])
        area = area + ta/2
       }
      } else if ( i == 1) {
       if (conc[i+1] == -1) {
        tempr = rbind(trilha$coordenadas[i:(i+1),],stop[(i+1)*2-2,],top[i,],trilha$coordenadas[i,])
        ta <- sum(tempr[1:(nrow(tempr)-1), 1] * tempr[2:nrow(tempr), 2]) -
sum(tempr[1:(nrow(tempr)-1), 2] * tempr[2:nrow(tempr), 1])
        area = area + ta/2 + ( dist(rbind(trilha$coordenadas[i,],top[i,]))^2 * ang(rbind(stop[(i+1)*2-
2,],trilha$coordenadas[i+1,],stop[(i+1)*2-1,]))[2] )/2
       } else {
        tempr = rbind(trilha$coordenadas[i:(i+1),],top[(i+1):i,],trilha$coordenadas[i,])
        ta <- sum(tempr[1:(nrow(tempr)-1), 1] * tempr[2:nrow(tempr), 2]) -
sum(tempr[1:(nrow(tempr)-1), 2] * tempr[2:nrow(tempr), 1])
        area = area + ta/2
       }
      } else if (i == (nrow(trilha$coordenadas) - 1) ) {
       if (conc[i] == -1) {
        tempr = rbind(trilha$coordenadas[i:(i+1),],top[i+1,], stop[i*2-1,],trilha$coordenadas[i,])
        ta <- sum(tempr[1:(nrow(tempr)-1), 1] * tempr[2:nrow(tempr), 2]) -
sum(tempr[1:(nrow(tempr)-1), 2] * tempr[2:nrow(tempr), 1])
        area = area + ta/2
       } else {
        tempr = rbind(trilha$coordenadas[i:(i+1),],top[(i+1):i,], trilha$coordenadas[i,])
        ta <- sum(tempr[1:(nrow(tempr)-1), 1] * tempr[2:nrow(tempr), 2]) -
sum(tempr[1:(nrow(tempr)-1), 2] * tempr[2:nrow(tempr), 1])
        area = area + ta/2
       }
      }
    } else if (d == 'bot') {
     if ( i > 1 && i < (nrow(trilha$coordenadas) - 1) ) {
      if (conc[i] == -1 && conc[i+1] == -1 ) { # se o proximo e o atual são concavos
        tempr = rbind(bot[i:(i+1),], trilha$coordenadas[(i+1):i,], bot[i,])
        ta <- sum(tempr[1:(nrow(tempr)-1), 1] * tempr[2:nrow(tempr), 2]) -
sum(tempr[1:(nrow(tempr)-1), 2] * tempr[2:nrow(tempr), 1])
        area = area + ta/2
       } else if (conc[i] == -1 && conc[i+1] > -1) { # se o proximo não é concavo mas o atual é
        tempr = rbind(bot[i,], sbot[(i+1)*2-2,],trilha$coordenadas[(i+1):i,],bot[i,])
        ta <- sum(tempr[1:(nrow(tempr)-1), 1] * tempr[2:nrow(tempr), 2]) -
sum(tempr[1:(nrow(tempr)-1), 2] * tempr[2:nrow(tempr), 1])
        area = area + ta/2 + ( dist(rbind(trilha$coordenadas[i,],bot[i,]))^2 * ang(rbind(sbot[(i+1)*2-
2,],trilha$coordenadas[i+1,],sbot[(i+1)*2-1,]))[2] )/2
       } else if (conc[i] > -1 && conc[i+1] == -1 ) { #se o proximo é concavo mas o atual não
        tempr = rbind(sbot[i*2-1,],bot[i+1,],trilha$coordenadas[(i+1):i,],sbot[i*2-1,])
        ta <- sum(tempr[1:(nrow(tempr)-1), 1] * tempr[2:nrow(tempr), 2]) -
sum(tempr[1:(nrow(tempr)-1), 2] * tempr[2:nrow(tempr), 1])
        area = area + ta/2
       } else { # se nenhum deles é concavo
        tempr = rbind(sbot[i*2-1,], sbot[(i+1)*2-2,], trilha$coordenadas[(i+1):i,], sbot[i*2-1,])
```

```
          ta <- sum(tempr[1:(nrow(tempr)-1), 1] * tempr[2:nrow(tempr), 2]) -
sum(tempr[1:(nrow(tempr)-1), 2] * tempr[2:nrow(tempr), 1])
          area = area + ta/2 + ( dist(rbind(trilha$coordenadas[i,],top[i,]))^2 * ang(rbind(sbot[(i+1)*2-
2,],trilha$coordenadas[i+1,],sbot[(i+1)*2-1,]))[2] )/2
         }
        }else if ( i == 1) {
         if (conc[i+1] == -1) {
          tempr = rbind(bot[i:(i+1),], trilha$coordenadas[(i+1):i,], bot[i,])
          ta <- sum(tempr[1:(nrow(tempr)-1), 1] * tempr[2:nrow(tempr), 2]) -
sum(tempr[1:(nrow(tempr)-1), 2] * tempr[2:nrow(tempr), 1])
          area = area + ta/2
         } else {
          tempr = rbind(bot[i,], sbot[(i+1)*2-2,], trilha$coordenadas[(i+1):i,], bot[i,])
          ta <- sum(tempr[1:(nrow(tempr)-1), 1] * tempr[2:nrow(tempr), 2]) -
sum(tempr[1:(nrow(tempr)-1), 2] * tempr[2:nrow(tempr), 1])
          area = area + ta/2 + ( dist(rbind(trilha$coordenadas[i,],top[i,]))^2 * ang(rbind(sbot[(i+1)*2-
2,],trilha$coordenadas[i+1,],sbot[(i+1)*2-1,]))[2] )/2
         }
        }else if (i == (nrow(trilha$coordenadas) - 1) ) {
         if (conc[i] == -1) {
          tempr = rbind(bot[i:(i+1),], trilha$coordenadas[(i+1):i,], bot[i,])
          ta <- sum(tempr[1:(nrow(tempr)-1), 1] * tempr[2:nrow(tempr), 2]) -
sum(tempr[1:(nrow(tempr)-1), 2] * tempr[2:nrow(tempr), 1])
          area = area + ta/2
         } else {
          tempr = rbind(sbot[i*2-1,], bot[i+1,], trilha$coordenadas[(i+1):i,], sbot[i*2-1,])
          ta <- sum(tempr[1:(nrow(tempr)-1), 1] * tempr[2:nrow(tempr), 2]) -
sum(tempr[1:(nrow(tempr)-1), 2] * tempr[2:nrow(tempr), 1])
          area = area + ta/2
         }
        }
      }
    }
  }
  area
}

# Cria o gráfico da parcela parâmetros semelhantes a calcArea incluindo o resultado de calcArea no
parâmetro <area>
plotParcela <- function(parcela, top, bot, lessAngle, nparcela, d = 'both', area) {
  plot(NULL, xlim = range(c(top$x,bot$x)), ylim = range(c(top$y,bot$y)), main =
paste(paste(nparcela,':',sep=''), round(area, 2)), axes = F, xlab = '', ylab = '')
  box()
  P <- angLess(parcela$coordenadas, lessAngle)
  for (i in 1:(nrow(parcela$coordenadas) - 1)) {
    if (!all(P[i:(i+1)] == T) && parcela$remove[i] == 0 ) {
      lines(parcela$coordenadas[i:(i+1),], col='red')
      if (d == 'both') {
        lines(top[i:(i+1),])
        lines(bot[i:(i+1),])
      } else if (d == 'top') {
```

```
      lines(top[i:(i+1),])
    } else if (d == 'bot') {
      lines(bot[i:(i+1),])
    }
  }
 }
}

# <data> tabela de dados (veja os comentários em doCentraline)
# <width> largura de cada um dos lados da parcela
# <lessAngle> ângulo (graus) de corta para remoção do segmento
# <d> se for 'top' calcula a área apenas para o lado 'norte', se se for 'bot'
#          calcula a área apenas para o lado 'sul' e se for 'both' para ambos os lados
#          <plotting> plotar as áreas? ([T]RUE, [F]ALSE)

doArea_i <- function(data, width = 5, lessAngle = 70, d = 'both', plotting = FALSE) {
 central <- by(data, data[, 1], doCentraline)
 area <- NULL
 for (i in 1:length(central)) {
  top <- bot <- data.frame(x = 0, y = 0)
  topp <- bott <- sut <- sub <- final <- data.frame(x = NA, y = NA)
  if (any(is.na(central[[i]]$angulos))) next
  for (j in 1:nrow(central[[i]]$coordenadas)) {
   top[j, 1] <- central[[i]]$coordenadas[j, 1] + width * cos(correctAng(central[[i]]$angulos[j] + pi / 2))
   top[j, 2] <- central[[i]]$coordenadas[j, 2] + width * sin(correctAng(central[[i]]$angulos[j] + pi / 2))
   bot[j, 1] <- central[[i]]$coordenadas[j, 1] + width * cos(correctAng(central[[i]]$angulos[j] - pi / 2))
   bot[j, 2] <- central[[i]]$coordenadas[j, 2] + width * sin(correctAng(central[[i]]$angulos[j] - pi / 2))
   if (j != nrow(central[[i]]$coordenadas)) {
    sut[j, 1] <- central[[i]]$coordenadas[j + 1, 1] + width * cos(correctAng(central[[i]]$angulos[j] + pi
/ 2))
    sut[j, 2] <- central[[i]]$coordenadas[j + 1, 2] + width * sin(correctAng(central[[i]]$angulos[j] + pi
/ 2))
    sub[j, 1] <- central[[i]]$coordenadas[j + 1, 1] + width * cos(correctAng(central[[i]]$angulos[j] - pi
/ 2))
    sub[j, 2] <- central[[i]]$coordenadas[j + 1, 2] + width * sin(correctAng(central[[i]]$angulos[j] - pi
/ 2))
   }
  }


  for (j in 1:(nrow(top) - 1)) {
   if (is.na(central[[i]]$angulos[j]) || is.na(central[[i]]$angulos[j+1]) ) break
   x <- central[[i]]$coordenadas[j + 1, ]
   x[1] <- x[1] + 10
   p1 <- ang( rbind(x, central[[i]]$coordenadas[j + 1, ], sut[j, ]) )[2]
   p2 <- ang( rbind(x, central[[i]]$coordenadas[j + 1, ], top[j + 1, ]) )[2]
   if ( is.na(p1) || is.na(p2)) {
    next
   }
   if ( sut[j, 2] < x[2] ) {
    p1 <- 2 * pi - p1
```

```
    }
    if ( top[j+1, 2] < x[2] ) {
      p2 <- 2 * pi - p2
    }
    mp <- mean(c(p1, p2))
    topp[j+1, ] <- c(central[[i]]$coordenadas[j + 1, 1] + width * cos(mp), central[[i]]$coordenadas[j +
1, 2] + width * sin(mp))
    bott[j + 1, ] <- c(central[[i]]$coordenadas[j + 1, 1] - width * cos(mp), central[[i]]$coordenadas[j +
1, 2] - width * sin(mp))
  }
  topp[1, ] <- top[1,]
  bott[1, ] <- bot[1,]
  for (j in 1:(nrow(topp) - 1) ) {
    if (findd(topp[j:(j + 1),],bott[j:(j + 1),]) ) {
      temp <- topp[j + 1,]
      topp[j + 1,] <- bott[j + 1,]
      bott[j + 1,] <- temp
    }
  }
  area[i] <- calcArea(central[[i]], topp, bott, sut, sub, lessAngle, d)
  if ( ! any(is.na(central[[i]]$angulos)) && plotting == T ) {
    plotParcela(central[[i]], topp, bott, lessAngle, names(central[i]), d, area[i])
  }

 }
 area
}

doArea <- function(topmin=0, botmin=0, ...) {
 args = list(...)
 area = 0
 width = args$width
 if (topmin > 0 && botmin > 0 ) {
  args$width = args$width + topmin
  args$d = "top"
  area = do.call(doArea_i, args)
  args$width = topmin
  areatop = area - do.call(doArea_i, args)

  args$width = width + botmin
  args$d = "bot"
  area = do.call(doArea_i, args)
  args$width = botmin
  areabot = area - do.call(doArea_i, args)

  area = areatop + areabot

 } else if (topmin > 0 && botmin == 0) {
  args$width = args$width + topmin
  args$d = "top"
  area = do.call(doArea_i, args)
```

```
  args$width = topmin
  area = area - do.call(doArea_i, args)
  args$d = "bot"
  args$width = width
  area = area + do.call(doArea_i, args)

 } else if (topmin == 0 && botmin > 0) {
  args$width = args$width + botmin
  args$d = "bot"
  area = do.call(doArea_i, args)
  args$width = botmin
  area = area - do.call(doArea_i, args)
  args$d = "top"
  args$width = width
  area = area + do.call(doArea_i, args)

 }
 if ( topmin == 0 && botmin == 0) {
  area = do.call(doArea_i, args)
 }
 area
}
```

##
##
################################## Até aqui...######################################
##
##

Aplicando a função

```
pdf('gráficos com as áreas.pdf')#Cria o gráfico em pdf
doArea(data = dados, plotting = TRUE)# Função que calcula área da parcela
dev.off()
```

As imagens podem ser encontradas na pasta de trabalho (digite: getwd( ) no R para saber sua atual pasta de trabalho)