Conferindo pares de coordenadas geográficas

Arquivo da função: Conf_coord

```
## Função "conf.coord" - por Default confere se um conjunto de pontos de coleta (pares de coordenadas geográficas) estão fora do continente e retorna o dataframe inicial de pares de coordenadas indicando quais estão dentro do continente, sugerindo correções. Opcionalmente, o usuário pode inserir um shapefile de área a ser conferida, neste caso, se os pontos estão DENTRO da área delimitada, ou não; também retornando o dataframe inicial de pares de coordenadas indicando quais estão fora da área, sugerindo correções. Como opção o usuário, a função faz um mapa para o usuário com os pontos plotados, destacando os que estão "errados".

## por: Giovana de Assis Garcia ##

conf.coord <- function(datafile, base=TRUE, fonte=NULL, mapa=FALSE)
{
  data.na <- na.omit(datafile) #cria dataframe de datafile sem NA.
  ## VERIFICAÇÕES DE PREMISSAS ##
  if (class(datafile) != "data.frame" | dim(datafile)[2] != 2) #se datafile não é dataframe ou se não tem 2 colunas
  {
    stop("Datafile deve ser um dataframe de 2 colunas.")
  }
  if (min(data.na[1]) < -90 | max(data.na[1]) > 90 | min(data.na[2]) < -180 | max(data.na[2]) > 180) #se datafile não está na ordem correta de colunas ou se está na formatação errada
  {
    stop("Datafile não está no formato correto, verifique no Help da função.")
  }
  if (base==FALSE & is.null(fonte))  #se base == FALSE e não é inserido arquivo em fonte para comparação
  {
    stop("Insira shapefile de região a ser comparada.")
  }
  if (base & !is.null(fonte)) # se base == TRUE mas é inserido arquivo em fonte, a função é executada como default de base==TRUE, imprime warning
  {
    warning("A comparação foi realizada com os continentes, de forma a identificar pontos que não estão no oceano. Para comparar com a região inserida no argumento fonte, use base==FALSE")
  }
  colnames(datafile) <- c("lat", "long") #padroniza nome das colunas do datafile
  ## DEFAULT => BASE DE COMPARAÇÃO É A LINHA DE COSTA MUNDIAL (BASE==TRUE) ##
  if (base)
```

```
  {
    #carrega pacotes necessários
    library(ocedata)
    library(sp)
    # criando dados da linha de costa: a ideia é ler latitude e longitude em
um dataframe, separar os continentes e fazer poligonos com eles. Os dados do
pacote Ocedata separam continentes por linha de "NA".
    data("coastlineWorldFine") #carregando dados
    coast <- coastlineWorldFine@data #guardando dados de lat e long em um
objeto
    coast$longitude <- as.numeric(coast$longitude) #deixando vetores
numéricos
    coast$latitude <- as.numeric(coast$latitude) #deixando vetores numéricos
    coast$latitude <- append(coast$latitude, NA) #adicionando NA no final
para "fechar" o ultimo continente
    coast$longitude <- append(coast$longitude, NA) #adicionando NA no final
para "fechar" o ultimo continente
    coast <- data.frame(lat = coast$latitude,long=coast$longitude)
#guardando em um dataframe
    vt.na <- which(is.na(coast$lat)) #vetor com posições de NA (que serão
utilizados como "separadores" de continentes)
    cont <- as.list(rep(NA,length(vt.na)-1)) #cria lista com número de
polígonos (=continentes)
    # neste laço para cada elemento na lista que representa os continentes,
serão guardados os pares de coordenadas, fazendo assim uma "lista de listas"
com todos os continentes. o vetor vt.na será utilizado como separador do
início e final de cada uma destas listas, dentro da lista.
    for (i in 1:length(cont))
    {
      cont[[i]] <- coast[seq(from=vt.na[i]+1, to=vt.na[i+1]-1),]
    }
    # é necessário criar uma matriz para comparar se os pontos estão dentro
ou fora dos continentes uma vez que a função "point.in.polygon" usa matrizes
    point <- matrix(NA, ncol=2,nrow=length(datafile$lat)) #criando matriz
com tamanho dos pontos a serem comparados
    point[,1] <- datafile$lat #atribuindo latitudes na primeira coluna da
matriz
    point[,2] <- datafile$long #atribuindo longitudes na segunda coluna da
matriz
    pontos <- rep(NA, times=length(datafile$lat)) #criando vetor que irá
"categorizar" os pontos entre: "dentro","fora" e "com possíveis erros", este
vetor será a base para todos os resultados
    # a lógica utilizada para comparaçaão com todos os continentes é:
comparar o ponto com os continentes e verificar qual a posição do ponto em
relação a cada continente. uma vez que os continentes aqui são vistos como
"Polígonos" para comparação, será criada uma matriz de dimensões [n
continentes, n pontos] que será responsável por armazenar a posição do ponto
em relação a cada um dos continentes (pois estar fora de um continente não
significa necessariamente estar no mar, pode-se encontrar dentro de outro
continente). Assim, posteriormente esta matriz irá definir a categoria final
```

```
dos pontos no vetor criado acima (pontos), de forma que se o ponto encontra-
se dentro de algum continente será categorizado como "dentro", ou se está
próximo ou na margem do continente será categorizado como "erro" e se não
está em nenhuma destas opções em nenhum dos continentes, este ponto com
certeza está no mar ("fora").
    #começando com a matriz que armazenará a posição relativa de cada ponto
a cada continente
    Mpontos <- matrix(NA, nrow=length(cont), ncol=length(pontos)) #criando
matriz de dimensões [n continentes, n pontos]
    for (j in 1:length(cont)) #criando laço que rodará todos os continentes
    {
      lat.pol <- as.matrix(cont[[j]]$lat) #transformando latitude em matriz
para poder comparar com a função "point.in.polygon"
      long.pol <- as.matrix(cont[[j]]$long) #transformando longitude em
matriz para poder comparar com a função "point.in.polygon"
      for (i in 1:length(datafile$lat)) #criando laço que rodará todos os
pontos a serem comparados
      {
        n <- point.in.polygon(point[i,1], point[i,2], lat.pol, long.pol)
#armazenando a posição relativa do ponto i no continente j
        #classificando a posição do ponto i relativa ao continente j a
partir do n
        if (n == 0)
        {
          Mpontos[j,i] <- "fora" #pontos fora do polígono (no caso,
continente)
        }
        if (n==1)
        {
          Mpontos[j,i] <- "dentro" #pontos totalmente dentro do polígono (no
caso, continente)
        }
        if (n==2 | n==3)
        {
          Mpontos[j,i] <- "erro" #pontos próximos ou na margem dos polígonos
(no caso, continentes)
        }
      }
    }
    #agora a função irá categorizar a posição do ponto relativa a TODOS os
continentes
    #serão criados contadores de pontos em cada categoria
    dentro = 0 #contador de pontos dentro
    fora = 0 #contador de pontos fora
    erro = 0 #contador de pontos com possíveis erros
    for (k in 1:length(pontos)) #criando laço que roda todos os pontos
    {
      if (is.element("dentro", Mpontos[,k])) #categorizando pontos que estão
dentro de algum continente
      {
        pontos[k] <- "dentro"
```

```
        dentro = dentro + 1
      }
      else
      {
        if (is.element("erro", Mpontos[,k])) #categorizando pontos que estão
próximos ou na margem de algum continente
        {
          pontos[k] <- "erro"
          erro = erro + 1
        }
        else
        {
          pontos[k] <- "fora"  #categorizando pontos que com certeza estão
no oceano
          fora = fora + 1
        }
      }
    }
    # impressões de resultados
    cat("Do total de", length(pontos), "pontos, foram encontrados:\n", fora,
"pontos no oceano\n", dentro, "pontos dentro dos continentes\n", erro,
"pontos com possíveis erros\n\n\n")
    if (dentro > 0)
    {
      coord.erradas <- datafile[which(pontos=="dentro"),] #cria dataframe
com pontos errados
      coord.erradas$pontos <- which(pontos=="dentro") #cria coluna com
numero dos pontos errados
    }
    if (erro > 0)
    {
      cat("Os pontos com possíveis erros são:", which(pontos=="possível
erro")) #imprime os pontos com possíveis erros
    }
  }
  ## SE BASE == FALSE ##
  # nesta opção, a principal comparação será se os pontos inseridos pelo
usuário estão dentro ou não da(s) região(ões) inseridas no shapefile
(diferentemente do default que compara principalmente se os pontos inseridos
estão fora do oceano).
  if (base==FALSE)
  {
    #carregando pacotes necessários
    library(sp)
    library(shapefiles)
    f <- read.shapefile(fonte) #lendo o shapefile inserido em "fonte"
    sh <- f[["shp"]][["shp"]] #selecionando apenas o shapefile que contém
pares de coordenadas das regiões a serem comparadas
    cont <- as.list(rep(NA,length(sh))) #criando lista que conterá os pares
de coordenadas de todas as regiões do shapefile
```

```
    for (i in 1:length(sh))
    {
      cont[[i]] <- sh[[i]][["points"]] #atribui a cada posição "i" da lista
"cont" o dataframe com os pares de coordenadas
      colnames(cont[[i]]) <- c("long","lat") #padroniza o nome das colunas
    }
    # assim como anteriormente, é necessário criar uma matriz para comparar
se os pontos estão dentro ou fora da(s) região(ões) uma vez que a função
"point.in.polygon" usa matrizes
    points <- matrix(NA, ncol=2,nrow=length(datafile$lat)) #criando matriz
com tamanho dos pontos a serem comparados
    points[,1] <- datafile$lat #atribuindo latitudes na primeira coluna da
matriz
    points[,2] <- datafile$long #atribuindo longitudes na segunda coluna da
matriz
    pontos <- rep(NA, times=length(datafile$lat)) #criando vetor que irá
"categorizar" os pontos entre: "dentro","fora" e "com possíveis erros", este
vetor será a base para todos os resultados
    # a lógica utilizada para comparaçaão com todas as regiões é a mesma que
no default: comparar o ponto com as regiões e verificar qual a posição do
ponto em relação a cada uma. uma vez que as regiões aqui são vistas como
"Polígonos" para comparação, será criada uma matriz de dimensões [n regiões,
n pontos] que será responsável por armazenar a posição do ponto em relação a
cada uma das regiões (pois estar fora de uma não significa necessariamente
estar errada, pode-se encontrar dentro de outra região). Assim,
posteriormente esta matriz irá definir a categoria final dos pontos no vetor
criado acima (pontos), de forma que se o ponto encontra-se fora de alguma
região será categorizado como "fora", ou se está próximo ou na margem de
alguma região será categorizado como "erro" e se não está em nenhuma destas
opções, este ponto com certeza está na região correta ("dentro").
    #começando com a matriz que armazenará a posição relativa de cada ponto
a cada região
    Mpontos <- matrix(NA, nrow=length(cont), ncol=length(pontos)) #criando
matriz de dimensões [n regiões, n pontos]
    for (j in 1:length(cont)) #criando laço que rodará todos as regiões
    {
      lat.pol <- as.matrix(cont[[j]]$lat) #transformando latitude em matriz
para poder comparar com a função "point.in.polygon"
      long.pol <- as.matrix(cont[[j]]$long) #transformando longitude em
matriz para poder comparar com a função "point.in.polygon"
      for (i in 1:length(datafile$lat)) #criando laço que rodará todos os
pontos a serem comparados
      {
        n <- point.in.polygon(points[i,1], points[i,2], lat.pol, long.pol)
#armazenando a posição relativa do ponto i na região j
        #classificando a posição do ponto i relativa ao continente j a
partir do n
        if (n == 0)
        {
          Mpontos[j,i] <- "fora" #classificando pontos fora do polígono
        }
```

```
        if (n==1)
        {
          Mpontos[j,i] <- "dentro" #classificando pontos dentro do polígono
        }
        if (n==2 | n==3)
        {
          Mpontos[j,i] <- "erro" #classificando pontos próximos ou na margem
do polígono
        }
      }
    }
    #agora a função irá categorizar a posição do ponto relativa a TODOS as
regiões
    #serão criados contadores de pontos em cada categoria
    dentro = 0 #contador de pontos dentro
    fora = 0 #contador de pontos fora
    erro = 0 #contador de pontos com possíveis erros
    for (k in 1:length(pontos)) #criando laço que roda todos os pontos
    {
      if (is.element("dentro", Mpontos[,k])) #categorizando pontos que estão
dentro de alguma região
      {
        pontos[k] <- "dentro"
        dentro = dentro + 1
      }
      else
      {
        if (is.element("erro", Mpontos[,k])) #categorizando pontos que estão
próximos ou na margem de alguma região
        {
          pontos[k] <- "erro"
          erro = erro + 1
        }
        else
        {
          pontos[k] <- "fora"  #categorizando pontos que com certeza estão
fora da região correta
          fora = fora + 1
        }
      }
    }
    # impressões de resultados
    cat("Do total de", length(pontos), "pontos, foram encontrados:\n",
dentro, "corretos \n", fora, "pontos fora da região inserida\n", erro,
"pontos com possíveis erros\n\n\n")
    if (fora > 0)
    {
      coord.erradas <- datafile[which(pontos=="fora"),] #cria dataframe com
os pares errados
      coord.erradas$pontos <- which(pontos=="fora") #cria coluna com numero
```

```
dos pontos errados
    }
    if (erro > 0)
    {
      cat("Os pontos com possíveis erros são:", which(pontos=="erro"))
#imprime quais os pontos com possíveis erros
    }
  }
  ## SE MAPA == TRUE ##
  if (mapa)
  {
    library(oce) #carregando pacote para plotar os mapas
    #definindo limites do mapa
    latlim <- c(min(datafile$lat)-
diff(range(datafile$lat)),max(datafile$lat)+diff(range(datafile$lat)))
#limite de latitude
    longlim <- c(min(datafile$long)-
diff(range(datafile$long)),max(datafile$long)+diff(range(datafile$long)))
#limite de longitude
    ## agrupando pontos de cada categoria para os plots ##
    #pontos fora do continente ou região
    latfora <- datafile$lat[which(pontos=="fora")]
    longfora <- datafile$long[which(pontos=="fora")]
    #pontos com possíveis erros
    laterro <- datafile$lat[which(pontos=="erro")]
    longerro <- datafile$long[which(pontos=="erro")]
    #pontos dentro do continente ou da região
    latdentro <- datafile$lat[which(pontos=="dentro")]
    longdentro <- datafile$long[which(pontos=="dentro")]
    x11() #criando device para plot
    ## para BASE == TRUE (comparação com continente) ##
    if (base)
    {
      mapPlot(coastlineWorldFine , latitudelim=latlim ,
longitudelim=longlim)  #plotando linha de costa com os limites relacionados
aos pontos
      mapPoints(latitude = latfora  , longitude = longfora, col="black",
pch=19) #plotando pontos corretos -> fora do continente (no oceano)
      mapPoints(latitude = latdentro, longitude = longdentro, col="red",
pch=13, cex=1.5) #plotando pontos errados -> dentro do continente
    }
    ## para BASE == FALSE (comparação com região inserida em fonte) ##
    else
      # neste caso precisamos criar vetores de TODAS as latitudes e
longitudes de TODAS as "microregiões" que pertencem a região inserida na
fonte
    {
      lines = 0 #contador do número total de pares de coordenadas na fonte
      for (i in 1:length(cont)) #laço que conta pares de coordenadas na
fonte
      {
```

```
        lines = lines + dim(cont[[i]])[1]
      }
      #criando vetores que conterão TODAS as latitudes e longitudes, e
portanto de tamanho igual ao contador criado (lines)
      latdata <- rep(NA, times=lines)
      longdata <- rep(NA, times=lines)
      #laço que atribuirá as latitudes e longitudes aos vetores criados
      l=1 #contador do indexador dos vetores latdata e longdata
      for (i in 1:length(cont)) #laço que roda todas as "microregiões" da
região
      {
        for (j in 1:length(cont[[i]]$lat)) #laço que roda todas as linhas do
dataframe de pares de coordenadas
        {
          latdata[l] <- cont[[i]]$lat[j] #atribuindo na posição "l" do
vetor, a latitude "j" referente ao continente "i"
          longdata[l] <- cont[[i]]$long[j] #atribuindo na posição "l" do
vetor, a longitude "j" referente ao continente "i"
          l=l+1 #preparando contador para a próxima posição dos vetores
        }
      }
      mapPlot(latitude=latdata, longitude=longdata , latitudelim=latlim ,
longitudelim=longlim, type="p", pch=".")  #plotando o contorno da região
dentro dos limites dos pontos
      mapPoints(latitude = latfora  , longitude = longfora, col="red",
pch=13, cex=1.5) #plotando pontos errados -> fora da região
      mapPoints(latitude = latdentro, longitude = longdentro, col="black",
pch=19) #plotando pontos corretos -> dentro da região
    }
    mapPoints(latitude = laterro  , longitude=longerro, col="gray", pch=19)
#plotando pontos de possíveis erros (fora do laço pois BASE não difere o
design destes pontos)
  }
  #retorna dataframe com os pontos errados, se existirem
  if (base==TRUE & dentro>0)
  {
  return(coord.erradas)
  }
  if(base==FALSE & fora>0)
  {
  return(coord.erradas)
  }
}
```