Curso de Análise de Dados Geográficos com R
Dezembro de 2015

SIGs com R: análise espacial

Manuel Campagnolo

Instituto Superior de Agronomia, Universidade de Lisboa

1. **Análise espacial em R**
2. **Métodos sobre objectos geométricos**
3. **Métodos para testar relações espaciais entre objectos geométricos**
4. **Métodos que suportam análise espacial**
5. **Análise de dados raster com package `sp`**

Conjuntos de dados geográficos vectoriais em R: análise espacial com o package `rgeos`

Dados vectoriais: conceitos fundamentais de análise espacial

O *OGC OpenGIS Implementation Standard for Geographic information* / ISO 19125 define métodos para **objectos geométricos**:

Objectos geométricos do tipo point, line, polygon, multi-point, etc, associados a um sistema de coordenadas;

Métodos sobre objectos geométricos devolvem propriedades tais como dimensão, fronteira, área, centroide, etc (ver Slide 5);

Métodos para testar relações espaciais entre objectos geométricos equals, disjoint, intersects, touches, crosses, within, contains, overlaps and relate, com valor lógico **TRUE** ou **FALSE** (ver Slide 13);

Métodos que suportam análise espacial distância, que devolve uma **distância**, e buffer, convex hull, intersection, union, difference, symmetric difference, que devolvem **novos objectos geométricos** (ver Slide 22).

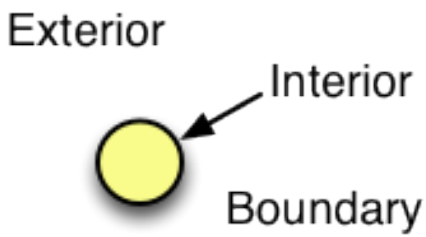
Fonte: `www.opengeospatial.org/standards/sfa`

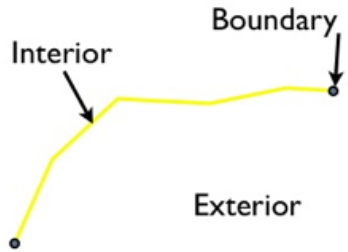
Métodos sobre objectos geométricos

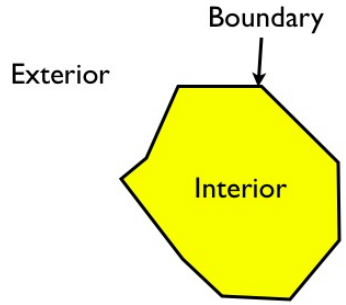
Objectos do tipo “ponto” (**p**) têm dimensão 0,
Objectos do tipo “linha” (**L**) têm dimensão 1,
Objectos do tipo “polígono” (**P**) têm dimensão 2

Os objectos geométricos têm **I**nterior, **F**ronteira e **E**xterior:

1. Se **p** é do tipo ponto, então **I**(**p**) = **p**, **F**(**p**) é vazio, e **E**(**p**) são todos os pontos que não estão em **p**;
2. Se **L** é do tipo linha, então **F**(**L**) são os extremos da linha, **I**(**L**) são todos os pontos da linha excepto os extremos, e o exterior **E**(**L**) são os todos os outros pontos;
3. Se **P** é do tipo polígono, **F**(**P**)é a fronteira, **I**(**P**) é o conjunto de pontos de **P** que não estão na fronteira e **E**(**P**) são os restantes pontos.







Análise espacial sobre objectos geométricos: o package `rgeos`

O package `rgeos` contém funções para implementar todos os métodos da norma descrita atrás. Para exemplificar vamos usar dados analisados anteriormente (dados do ICNF):

```
setwd(paste0(getwd(),"\\dados_curso")) # definir pasta de
    trabalho
library(raster)
library(sp)
library(rgdal) # para importar e exportar dados
library(rgeos) # para análise espacial
icnf<-readOGR(dsn=getwd(),layer="AP_JUL_2014",encoding="ISO8859
    -1")
```

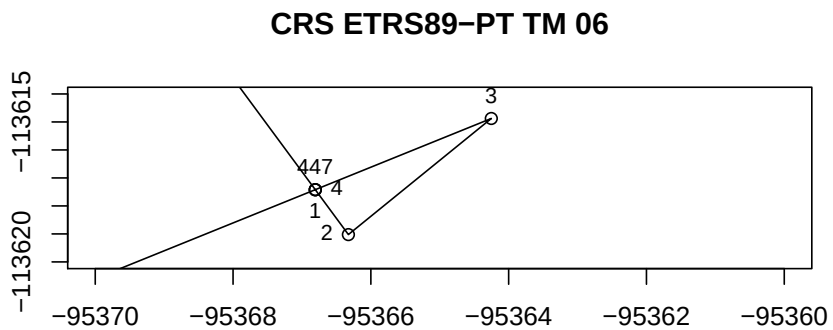
Para usar funções do package rgeos, as geometrias dos objectos devem ser válidas; a função `gIsvalid` do package `rgeos` permite saber quais são os objectos geométricos válidos:

```
valid<-gIsValid(icnf,byid=TRUE,reason=TRUE)
which(valid!="Valid Geometry") #  13 55 58 61 inválidos
```

Análise espacial sobre objectos geométricos: o package `rgeos`: ferramentas de validação da topologia

O output de `valid` permite perceber onde ocorre o problema:

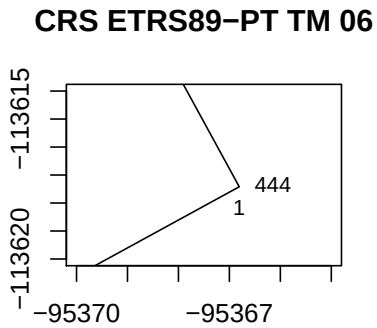
```
valid[13]  # "Ring Self-intersection[-95366.8066 -113617.4198]"
plot(icnf[13,]) # Arriba Fossil da Costa da Caparica
zoom(icnf[13,],ext=extent(c(-95370,-95360, -113620,-113610)),
    new=FALSE)
# para ver as coordenadas dos vértices:
xy<-icnf[13,]@polygons[[1]]@Polygons[[1]]@coords
text(xy,apply(round(xy),1,paste,collapse=","))
# para ver os indices dos vértices
zoom(icnf[13,],ext=extent(c(-95370,-95360, -113620,-113610)),
    new=FALSE)
text(xy,labels=1:nrow(xy),pos=c(1,2,3,4),cex=.8) # o problema é
     nos vértices 2, 3 e 4
```



Análise espacial sobre objectos geométricos: o package `rgeos`: ferramentas de validação da topologia

Eliminar os vértices 2, 3 e 4 e reconstruir feature 13:

```
1 icnf.novo<-icnf
2 part1 <- Polygon(xy[-c(2,3,4),],hole=FALSE
      )
3 # icnf[13,] tem uma única multiparte que
      vamos subtituir por feature13:
4 feature13 <- Polygons(list(part1),row.
      names(icnf)[13]) # criar objecto
      multiparte
5 icnf.novo@polygons[[13]]<-feature13
6 zoom(icnf.novo[13,],ext=extent(c
      (-95370,-95360, -113620,-113610)),new=
      FALSE)
```



Poder-se-ia corrigir da forma acima os problemas topológicos, mas vamos restringir a análise aos objectos geométricos válidos:

```
1 valid<-gIsValid(icnf.novo,byid=TRUE,reason=TRUE)
2 which(valid!="Valid Geometry") # feature 13 já não é inválida
```

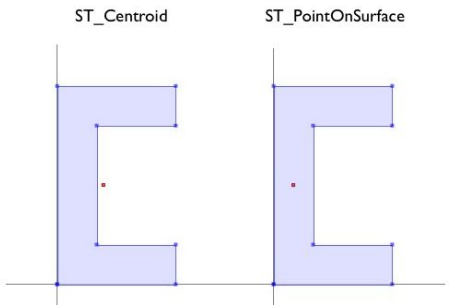
Conjuntos de dados geográficos vectoriais em R: Métodos sobre objectos geométricos

Algumas propriedades de objectos geométricos: área e controide

Os objectos geométricos de dimensão 2 têm área, que é devolvida pela função `gArea`.

A área é calculada relativamente às coordenadas dos vértices do objecto geométrico. Assim, uma transformação de coordenadas que cause distorções da geometria pode alterar a área determinada.

Outra propriedade básica de objectos geométricos é o centroide, isto é, o centro de gravidade dos vértices que delimitam o objecto. O centroide pode estar colocado fora do próprio objecto. Por essa razão o standard do OGC determina a existência de um método (`PointOnSurface`) que devolve um ponto situado obrigatoriamente sobre o objecto. O package rgeos disponibiliza esse método através da função `gPointOnSurface`.



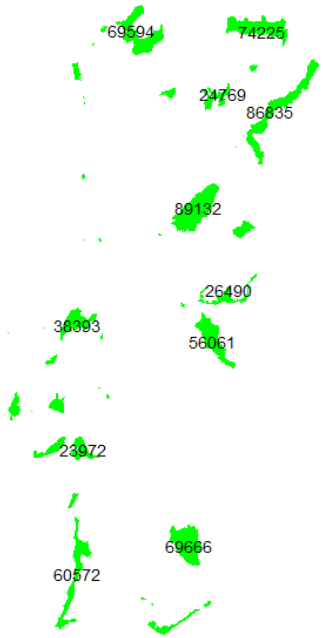
Análise espacial: métodos para objectos geométricos

Determinar áreas as áreas protegidas com a função `gArea`:

```
gArea(icnfv) # devolve a área total
gArea(icnfv,byid=TRUE) # áreas da totalidade de ''features''
```

Para determinar um ponto sobre as "features" usa-se `gPointOnSurface`:

```
plot(icnfv,col="green",border=FALSE)
areas <- round(gArea(icnfv,byid=TRUE
    )/10000) # areas em ha
xy <- coordinates(gPointOnSurface(
    icnfv,byid=TRUE))
text(xy[areas > 20000,1],xy[areas
    >20000,2],paste(areas[areas
    >20000]), cex=0.7)
```



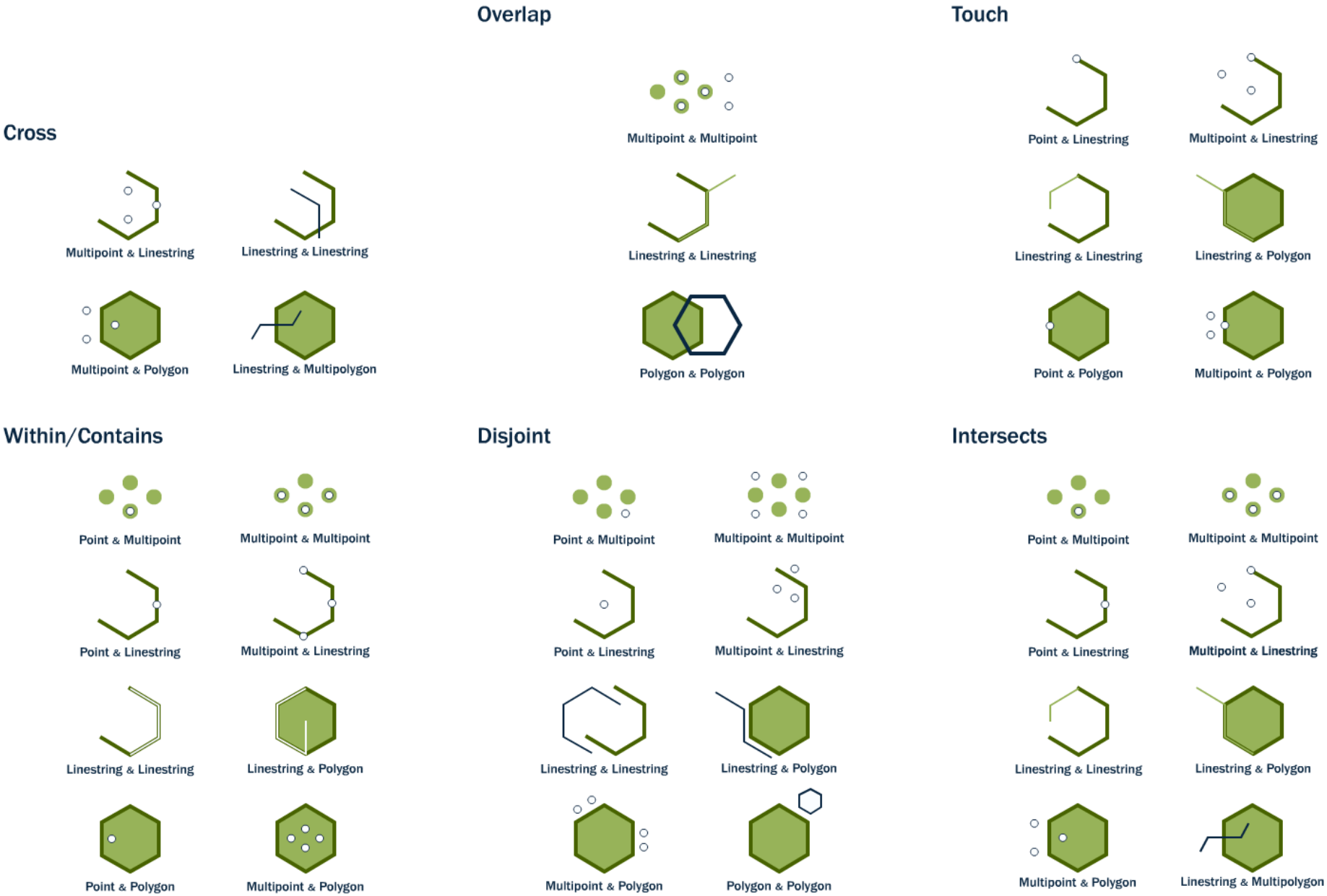
Conjuntos de dados geográficos vectoriais em R: Métodos para testar relações espaciais entre objectos geométricos

Métodos para testar relações espaciais entre objectos geométricos

A norma OGC/ISO (ver Slide 4) define um conjunto de operadores com valor lógico (**TRUE** ou **FALSE**) para testar relações espaciais entre objectos geométricos **p**onto, **L**inha, **P**olígono.

1. Equal: os objectos coincidem;
2. Disjoint: os objectos não têm pontos em comum;
3. Touches aplica-se a 2 objectos desde que algum tenha dimensão maior que 0: *a* Touches *b* se as fronteiras se intersectam;
4. Para *a*/*b* de tipo **p**/**L**, **p**/**P**, **L**/**L** or **L**/**P**, *a* Crosses *b* se os objectos se intersectam mas nenhum contém o outro;
5. *a* Within *b* se *a* não intersecta o exterior de *b*;
6. Overlaps aplica-se a objectos com mesma dimensão: é **TRUE** se os objectos se intersectam mas nenhum contém o outro;
7. *a* Contains *b* se *b* Within *a*;
8. *a* Intersects *b* se *a* e *b* têm algum ponto em comum;
9. Relates permite definir qualquer relação espacial.

Relações espaciais entre objectos geométricos (ilustração)



Métodos para testar relações espaciais entre objectos geométricos com package `rgeos`

Estes testes permitem selecções por localização. A ideia é seleccionar um subconjunto de objectos de uma "feature class" com base na relação espacial relativamente a outra "feature class".

No exemplo abaixo seleccionam-se "features" (concelhos) que intersectam as linhas do cdg `rios` que têm "Rio Mondego" na designação. Em primeiro vamos ler e pré-processar dados:

```
1 rios<-readOGR(dsn=getwd(),layer="rios",encoding="ISO8859-1")
2 concelhos<-readOGR(dsn=getwd(),layer="conc_2013",encoding="
      ISO8859-1") # cuidado com o CRS
3 igeoe <- "+proj=tmerc +lat_0=39.66666666666666 +towgs84
      =-283.1,-70.7,117.4,-1.16,0.06,-0.65,-4.1 +lon_0=1 +k=1 +x_
      0=200000 +y_0=300000 +ellps=intl +pm=lisbon +units=m"
4 concelhos@proj4string<-CRS(igeoe) # corrige CRS que estava
      incompleto
5 valid<-gIsValid(concelhos,byid=TRUE,reason=TRUE)
6 which(valid!="Valid Geometry") # devolve índice 88
7 concelhos[88,] # Ílhavo
8 concelhosv<-concelhos[valid=="Valid Geometry",]
```

Métodos para testar relações espaciais entre objectos geométricos com package `rgeos`

Re-projectar os cdg `rios` e `concelhosv` para o mesmo CRS que `icnfv` (que tem coordenadas ETRS89-PT TM 06):

```
1 rios.etrs<-spTransform(rios,icnfv@proj4string)
2 concelhos.etrs<-spTransform(concelhosv,icnfv@proj4string)
```

Seleccionar Mondego:

```
1 rio<-rios.etrs[grepl(pattern="(R|r)io.*(M|m)ondego",rios.
      etrs@data$DESIGNACAO),]
```

Métodos para testar relações espaciais entre objectos geométricos com package `rgeos`

Determinar concelhos que intersectam linhas de `rio`:

```
its<-gIntersects(concelhos.etrs,rio,byid=TRUE)
# its é uma matriz lógica (TRUE/FALSE) 48*277
# 48 features para rio e 277 para concelhos.etrs
dim(its)
```

Para selecionar os concelhos que são intersectados pelo rio é necessário saber quais as colunas da matriz `its` que têm pelo menos um `TRUE`:

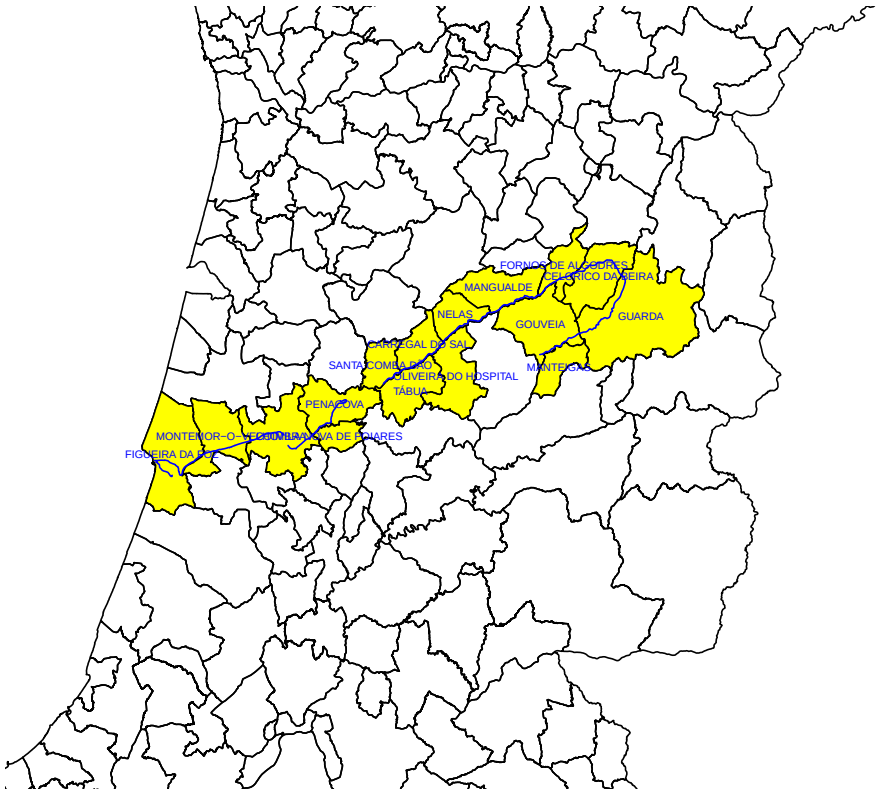
```
# definir vector de TRUE e FALSE para os concelhos
aux<-as.logical(apply(its,2,max))
```

Os concelhos seleccionados são `concelhos.etrs[aux,]`.

Métodos para testar relações espaciais entre objectos geométricos com package `rgeos`

Podemos então criar um mapa com esses concelhos:

```
1 plot(concelhos.etrs, xlim=c
      (-95000, 151000), ylim=c
      (11000,136000))
2 plot(concelhos.etrs[aux,],add=
      TRUE, col="yellow")
3 xy<-coordinates(concelhos.etrs[
      aux,])
4 text(xy[,1],xy[,2],labels=
      concelhos.etrs@data[aux,"
      Municipio"],cex=.5,col="
      blue")
5 plot(rio,add=TRUE,col="blue")
```



Métodos para testar relações espaciais entre objectos geométricos com package `rgeos`

Vamos agora identificar os concelhos no Parque da Serra da Estrela. Em primeiro lugar vamos seleccionar (selecção por atributos) esse parque:

```
parque<-icnfv[grepl(pattern='estrela',icnfv@data$NOME,ignore.
    case = TRUE),]
```

Seleccionar concelhos que intersectam o parque:

```
its<-gIntersects(concelhos.etrs,parque,byid=TRUE)
aux<-as.logical(apply(its,2,max)) # vector lógico
concelhos.etrs[aux,]
```

Seleccionar concelhos que que estão contidos no parque:

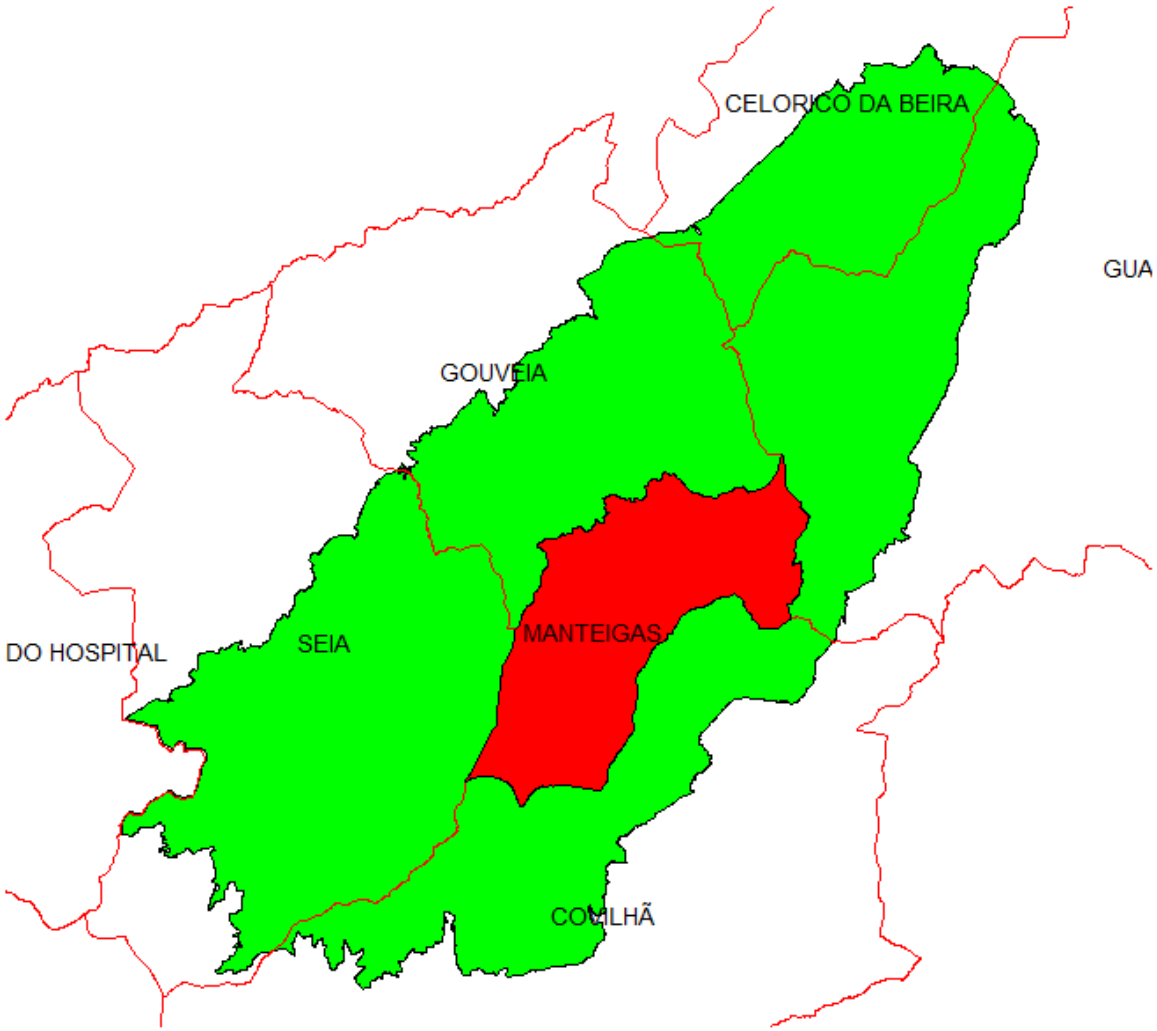
```
its<-gWithin(concelhos.etrs,parque,byid=TRUE)
aux<-as.logical(apply(its,2,max)) # vector lógico
concelhos.etrs[aux,]
```

Métodos para testar relações espaciais entre objectos geométricos com package `rgeos`

Para criar mapa:

```
plot(parque,col="green")
# concelhos que intersectam o Parque
its<-gIntersects(concelhos.etrs,parque,byid=TRUE)
aux<-as.logical(apply(its,2,max)) # vector logico
plot(concelhos.etrs[aux,],add=TRUE,border="red")
# escrever nomes dos concelhos seleccionados
xy<-coordinates(gPointOnSurface(concelhos.etrs[aux,],byid=TRUE)
    )
text(xy[,1],xy[,2],concelhos.etrs@data[aux,"Municipio"],cex
    =0.7)
# concelhos totalmente contidos no parque:
its<-gWithin(concelhos.etrs,parque,byid=TRUE)
aux<-as.logical(apply(its,2,max)) # vector logico
plot(concelhos.etrs[aux,],add=TRUE,col="red")
# escrever nome do concelho seleccionado
xy<-coordinates(gPointOnSurface(concelhos.etrs[aux,],byid=TRUE)
    )
text(xy[,1],xy[,2],concelhos.etrs@data[aux,"Municipio"],cex
    =0.7)
```

Métodos para testar relações espaciais entre objectos geométricos com package `rgeos`



Métodos que suportam análise espacial

A norma OGC/ISO (ver Slide 4) define operadores que devolvem a distância entre objectos geométricos ou novos objectos geométricos de acordo com alguma relação espacial. Dados objectos *a* e *b*:

1. Distance devolve a distância mais curta entre pontos de *a* e de *b*;
2. Buffer de *a* com distância *d* devolve o objecto geométrico cujos pontos estão uma distância menor ou igual a *d* de *a*;
3. ConvexHull de *a* devolve um objecto geométrico que é o invólucro convexo de *a*;
4. Intersection devolve o objecto geométrico cujos pontos estão em *a* **e** em *b*;
5. Union devolve o objecto geométrico cujos pontos estão em *a* **ou** em *b*;
6. Difference entre *a* e *b* devolve o objecto geométrico com os pontos que estão em *a* mas não em *b* (operação não comutativa);
7. Symmetric Difference entre *a* e *b* devolve o objecto geométrico cujos pontos satisfazem **apenas uma** das condições: (1) estão em *a* mas não em *b*; (2) estão em *b* mas não em *a*.

Conjuntos de dados geográficos vectoriais em R: Métodos que suportam análise espacial

Métodos que suportam análise espacial com package `rgeos`

A função `gDistance` permite calcular distâncias entre "features" de dois objectos `sp`. No exercício seguinte determinam-se as distâncias entre os concelhos e as áreas protegidas com área superior a 1000ha. O resultado da função é uma matriz de distâncias.

Como `gDistance` deve comparar todos os vértices que definem as "features" de um e outro conjunto de dados, **o tempo de processamento pode ser muito elevado**. Uma forma de reduzir esse tempo de processamento é considerar apenas os centroides das "features" e calcular as distâncias entre esses centroides.

Definir uma função que converte `SpatialPolygonsDataFrame` em `SpatialPointsDataFrame` em que os pontos do segundo são os centroides das "features" do primeiro:

```
1 simplifica<-function(spdf)
2 {
3   sp<-gCentroid(spdf,byid=TRUE) # devolve centroides das "
        features"
4   return(SpatialPointsDataFrame(sp,spdf@data))
5 }
```

Métodos que suportam análise espacial com package `rgeos`

Definir novo cdg com as áreas protegidas com área superior a 1000ha:

```
areas.ha <- round(as.vector(gArea(icnfv,byid=TRUE))/10000)
icnfg<-icnfv[areas.ha>1000,]
```

Em seguida calculam-se as distâncias; `d` é uma matriz 27×277 que contem as distâncias entre os centroides dos concelhos e as áreas protegidas com mais de 1000ha:

```
centroides.concelhos<-simplifica(concelhos.etrs)
d<-gDistance(centroides.concelhos,icnfg,byid=TRUE)
# atribuir nomes às linhas e colunas da matriz para facilitar a
    interpretação da mesma
colnames(d)<-as.character(centroides.concelhos$Municipio)
rownames(d)<-as.character(icnfg$NOME)
```

Métodos que suportam análise espacial com package `rgeos`

Qual é distância mínima (em Km) de cada concelho a uma área protegida com mais de 1000ha?

```
dc<-round(apply(d,2,min))
names(dc)<-as.character(centroides.concelhos$Municipio)
```

Qual o concelho que fica à maior distância?

```
dc[dc==max(dc)] # Mira, a 81 km
```

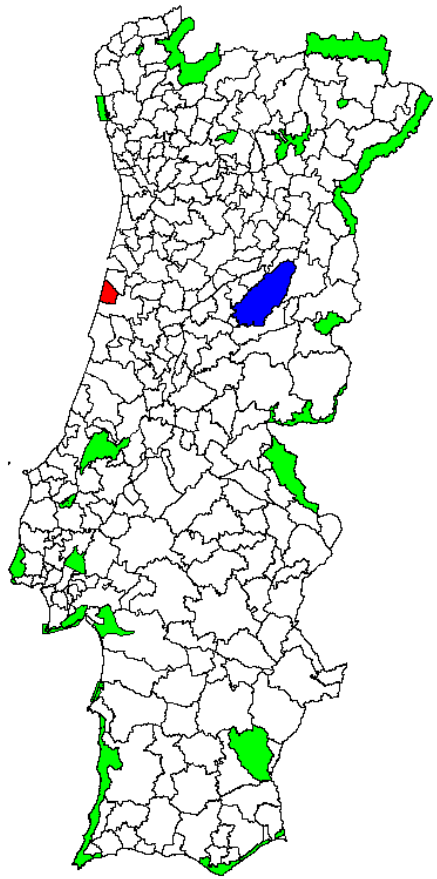
Qual é a área protegida mais próxima desse concelho

```
ind<-which(dc==max(dc)) # devolve o índice do concelho (coluna
    de d)
aux<-d[,ind] # distâncias desse concelho a todas as áreas
    protegidas
aux[aux==min(aux)] # devolve a área protegida à menor distância
     do concelho
```

Métodos que suportam análise espacial com package `rgeos`

Fazer figura em que as áreas protegidas com mais de 1000ha estão indicadas a verde, e o concelho que fica a maior distância de alguma área protegida (Mira) está desenhado a vermelho. A área protegida mais próxima de Mira (Serra da Estrela) está representada a azul.

```
1 plot(concelhos.etrs)
2 plot(concelhos.etrs[ind,], col="red",
      add=TRUE)
3 plot(icnfg, col="green",add=TRUE)
4 plot(icnfg[aux==min(aux),], col="blue
      ",add=TRUE)
```



Métodos que suportam análise espacial com package `rgeos`

A definição de *buffers* é uma operação muito usada em SIGs. Em `rgeos` a funçao respectiva é `gBuffer`.

Construir um buffer em torno da área protegida da Serra de Montejunto:

```
1 plot(gBuffer(icnfv[12,], byid=TRUE, width=100), border="red")
2 plot(icnfv[12,], add=TRUE)
```

Notas: O argumento `width` pode ser negativo. A função `gBuffer` aceita argumentos adicionais que são úteis sobretudo para buffers de linhas, como por exemplo,

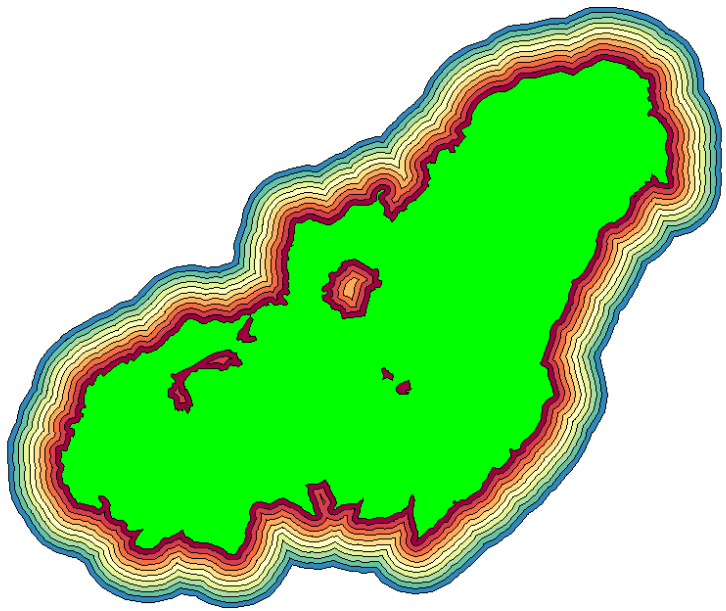
1. `quadsegs` que é o número de segmentos para aproximar um quarto de círculo;
2. `capStyle` que é a forma do buffer no extremo da linha (`ROUND`, `FLAT`, `SQUARE`).

Métodos que suportam análise espacial com package `rgeos`

Pode, obviamente, criar-se vários anéis de *buffer* com um ciclo. Neste caso usam-se cores do package abaixo para os anéis.

```
1 library(RColorBrewer) # para criar paletas
2 display.brewer.all() # ver possíveis paletas de cores
```

```
1 par(mar=rep(0,4))
2 plot(gBuffer(icnfv[12,],width=1100),
      border=FALSE)
3 for (i in 10:1) plot(gBuffer(icnfv
      [12,],byid=TRUE,width=100*i),add=
      TRUE,col=brewer.pal(n=11, "
      Spectral")[i])
4 plot(icnfv[12,],add=TRUE,col="green"
      )
```



Métodos que suportam análise espacial com package `rgeos`

A `dissolução` de "features" faz-se com a função `gUnaryUnion`.

Para exemplificar, criar `SpatialPolygonsDataFrame` dos distritos de Portugal Continental usando como atributo para agrupar concelhos o atributo `Distrito` da tabela de atributos de `concelhos.etrs`:

```
1 distritos<-gUnaryUnion(concelhos.etrs, id = concelhos.etrs$
    Distrito)
```

`gUnaryUnion` devolve um objecto `SpatialPolygons` sem tabela de atributos. Para obter `SpatialPolygonsDataFrame` é necessário criar a tabela de atributos e associá-la a `distritos`.

Os nomes das linhas dessa tabela de atributos terão que ser idênticos aos identificadores das *features* no objecto `distritos` que podemos obter fazendo:

```
1 names(distritos)
```

Métodos que suportam análise espacial com package `rgeos`

A função `aggregate()` permite agrupar linhas de uma `data.frame` e aplicar a cada um dos grupos uma determinada função. Neste caso, vamos agrupar os concelhos por distrito, e aplicar a função `sum` para obter as áreas dos distritos.

```
1 # determinar áreas dos concelhos com função area()
2 area(concelhos.etrs)
3 # substituir atributo area de concelhos por esses novos valores
4 concelhos.etrs@data$area<-area(concelhos.etrs)
5 # agregar as áreas dos concelhos do mesmo distrito:
6 aggregate(area~Distrito,data=concelhos.etrs,sum)
```

O resultado é uma `data.frame`.

Métodos que suportam análise espacial com package `rgeos`

Vamos então completar a criação da `SpatialPolygonsDataFrame` dos distritos.

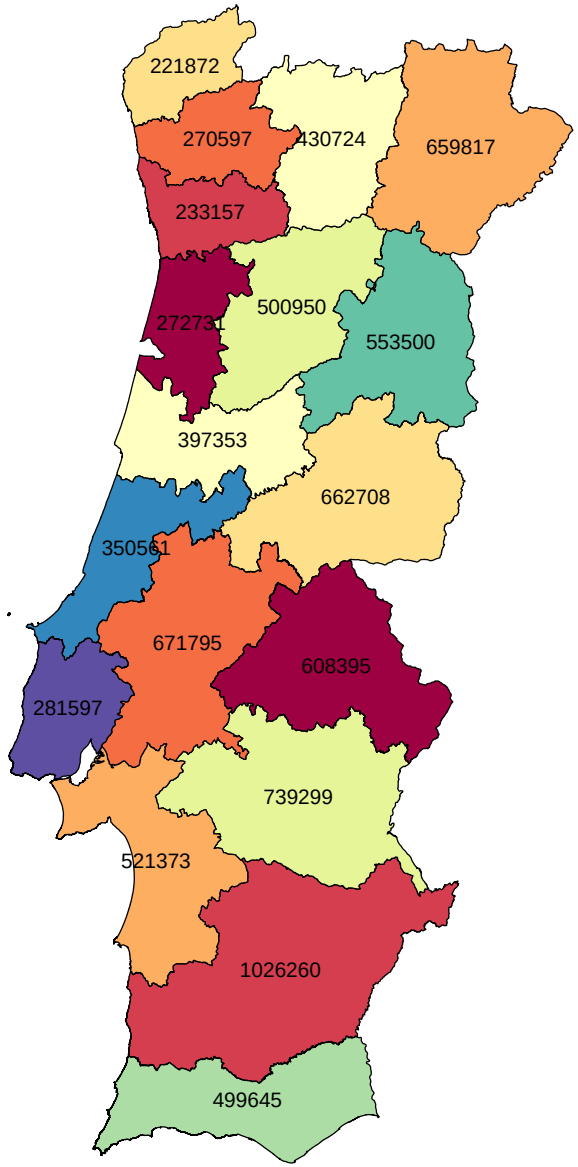
Como referido atrás, os nomes das linhas da tabela de atributos terão que ser os IDs associados às features de distritos (que são devolvidos por `names(distritos)`).

Vamos então atribuir os nomes correctos à linhas da `data.frame` criada com a função `aggregate` e associar essa tabela de atributos a distritos:

```
1 df.distritos<-data.frame(aggregate(area~Distrito,data=concelhos
      .etrs,sum), row.names=names(distritos))
2 # Definir SpatialPolygonsDataFrame
3 distritos.etrs<-SpatialPolygonsDataFrame(distritos,data=df.
      distritos)
```

Métodos que suportam análise espacial com package `rgeos`

```
1 plot(distritos.etrs,col=brewer.
      pal(n=11, "Spectral"))
2 # area em ha de cada distrito
3 xy<-as.matrix(coordinates(
      distritos.etrs))
4 text(xy[,1],xy[,2],round(
      distritos.etrs$area/10000))
```



Métodos que suportam análise espacial com package `rgeos`

A operação espacial de `intersecção` de "features" de objectos `sp` realiza-se com a função `gIntersection`. Voltando ao exemplo dos concelhos no Parque Natural da Serra da Estrela:

```
1 itsct<-gIntersection(parque,concelhos.etrs, byid=TRUE)
```

De novo, `itsct` é de classe `SpatialPolygons`. Para obter um objecto de classe `SpatialPolygonsDataFrame` é preciso definir a tabela de atributos. Pretende-se ter na tabela de atributos o nome do concelho, do distrito, e o nome do Parque.

Os IDs associados às "features" de `itsct` são:

```
1 names(itsct) # "34 108" "34 132" "34 134" "34 135" "34 138" "34
      170" "34 245"
```

Os IDs são definidos pela concatenação dos IDs de parque, neste caso sempre 34, e dos IDs de `concelhos.etrs`.

Métodos que suportam análise espacial com package `rgeos`

Para saber qual é o parque e qual é o concelho é preciso:

1. extrair o ID respectivo;
2. usar as tabelas de atributos de `parque` e `concelhos.etrs` para obter as designações.

A função `strsplit` separa e devolve uma lista com 7 componentes:

```
1 spl<-strsplit(names(itsct),split=" ")
2 spl.matrix<-matrix(unlist(spl),ncol=2,byrow=TRUE)
```

```
     [,1] [,2]
[1,] "34" "108"
[2,] "34" "132"
[3,] "34" "134"
[4,] "34" "135"
[5,] "34" "138"
[6,] "34" "170"
[7,] "34" "245"
```

Métodos que suportam análise espacial com package `rgeos`

Criar vector com os nomes das áreas protegidas que estão na intersecção definida:

```
df.icnf<-parque@data[spl.matrix[,1],c("NOME","CLASSIFICA")]
```

Proceder de igual forma para obter os nomes dos municípios e dos distritos correspondentes:

```
df.conc<-concelhos.etrs@data[spl.matrix[,2],c("Municipio","
    Distrito")]
```

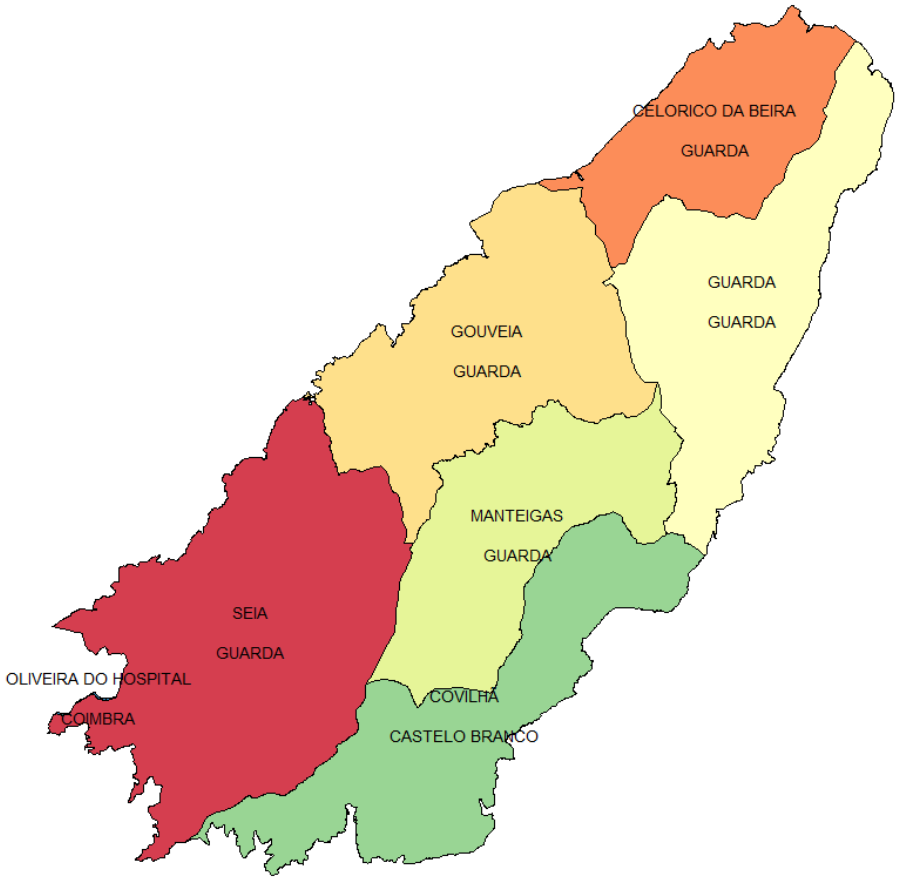
Finalmente, associar essa tabela a `itsct`:

```
df.itsct <- data.frame(cbind(df.icnf,df.conc),row.names=names(
    itsct))
itsct.etrs<-SpatialPolygonsDataFrame(itsct,data=df.itsct)
```

Métodos que suportam análise espacial com package `rgeos`

Criar imagem:

```
1 par(mar=rep(0,4))
2 plot(itsct.etrs,col=brewer.pal(
      n=length(itsct.etrs), "
      Spectral"))
3 # para as várias features:
4 for (i in 1:length(itsct.etrs))
5 {
6   xy<-itsct.etrs@polygons[[i]]
        @labpt
7   text(xy[1],xy[2],itsct.
        etrs@data[i,"Municipio"],
        pos=3,cex=0.6)
8   text(xy[1],xy[2],itsct.
        etrs@data[i,"Distrito"],
        pos=1,cex=0.6)
9 }
```



Métodos que suportam análise espacial com package `rgeos`

Parece haver um problema com a delimitação da fronteira do parque e /ou do concelho de Oliveira do Hospital. Vamos determinar a área da intersecção entre os dois:

```
1 itsct.etrs@data # para observar a tabela de atributos
2 int<-itsct.etrs[itsct.etrs$Municipio=="OLIVEIRA DO HOSPITAL" &
      itsct.etrs$NOME=="Serra da Estrela",]
3 gArea(int) # menos de 20 ha
```

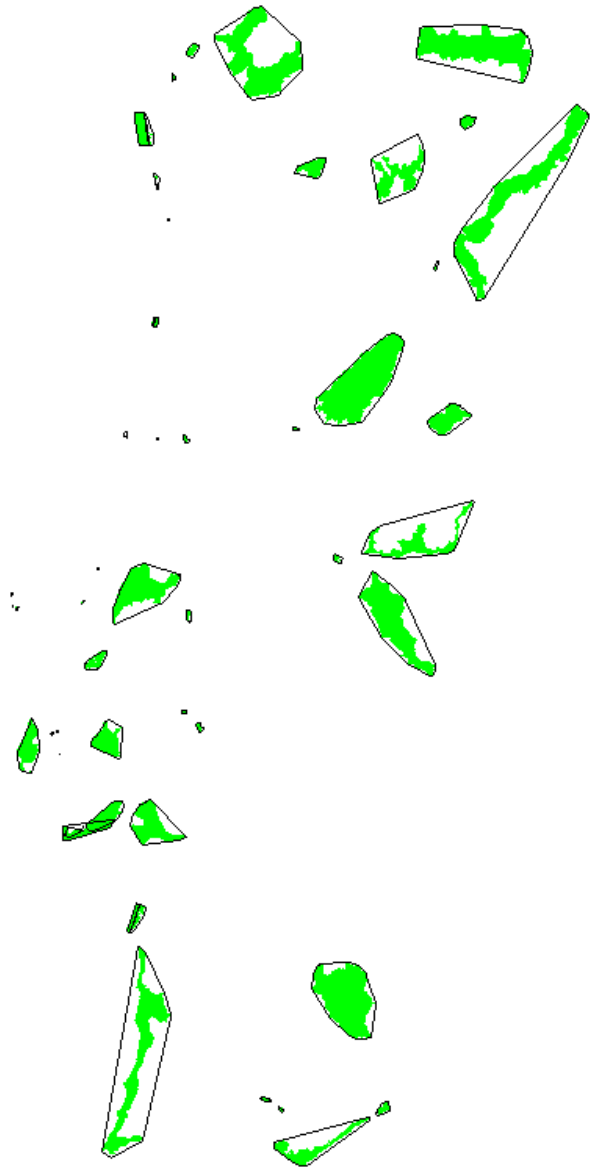
Se de facto a fronteira dos concelhos está correcta e se o parque natural da Serra da Estrela não deve intersectar o concelho de Oliveira do Hospital, pode retirar-se com a função gDifference a zona de intersecção ao conjunto de dados geográficos que representa o parque.

```
1 novoParque<-gDifference(parque,int,byid=TRUE)
2 plot(parque,xlim=c(27142, 33728), ylim=c(73244,68468))
3 plot(novoParque,col="red",add=TRUE)
```

Métodos que suportam análise espacial com package `rgeos`

A função `gConvexHull` devolve o invólucro convexo:

```
1 plot(icnfv,col="green",border=
      FALSE)
2 plot(gConvexHull(icnfv,byid=
      TRUE),add=TRUE)
```



Análise de dados raster com package `sp`

Dados matriciais: leitura e escrita com `rgdal`:

```
library(rgdal)
# ler ficheiro
g <- readGDAL("landsat8pan.tif")
```

`g` é um objecto da classe `SpatialGridDataFrame` (package `sp`).

```
summary(g) # descreve g
str(g) # devolve a estrutura de g
```

Os objectos das classes de `sp` têm os seguintes "slots":

1. `@bbox`: extensão geográfica dos dados;
2. `@proj4string`: sistema de coordenadas, de classe `CRS`;
3. `@data`: contém os valores associados às observações geográficas.

Para a classe `SpatialGridDataFrame`, há o "slot" adicional:

1. `@grid`: estrutura espacial dos dados, i.e. o tamanho (resolução) e coordenadas das células (ou pixels) do conjunto de dados matricial.

Dados matriciais: leitura e escrita com `rgdal`:

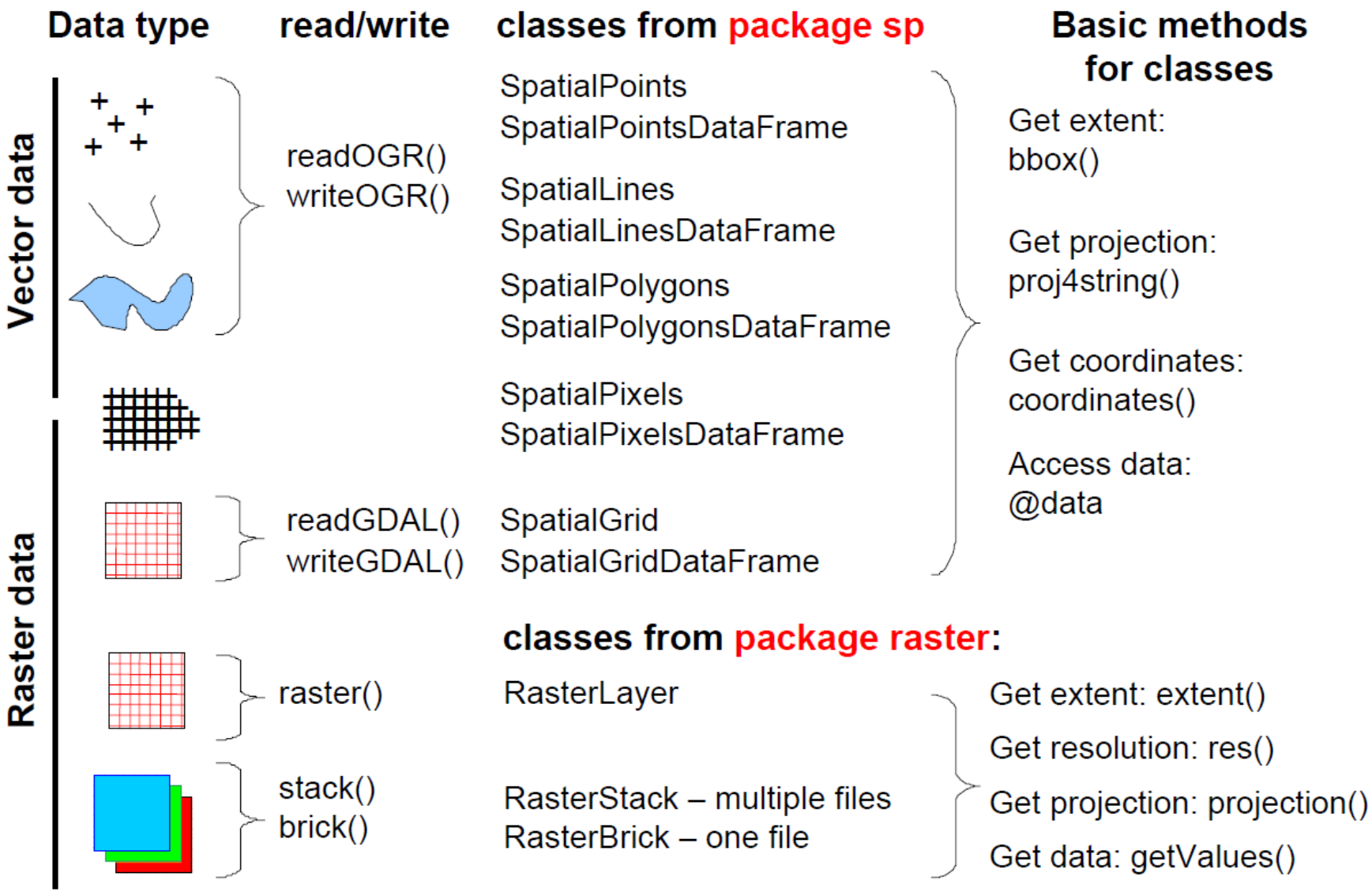
```
# g@data é uma das "slots" do objecto: é uma data.frame
# os nomes das colunas de g@data são então dados por
names(g@data) # band1
# histograma dos valores da imagem:
hist(g@data$band1)
cores <- gray(seq(0,1,length.out=100))
# image é melhor do que plot, que bloqueia para landsat8 pan
    (225*10^6 pixels)
image(g,col=cores,zlim=quantile(g@data$band1,probs=c(0,.99)))
```

Exportar o objecto `g` para um ficheiro GeoTIFF:

```
writeGDAL(g,fname="out.tif") #
```

Em jeito de conclusão

Classes for spatial data in R



Em jeito de conclusão (raster)

Alguns funções (ou slots) definidos pelo package `raster`:

1. Importar/exportar: `raster()`, `brick()` `stack()` (para imagens múltiplas) e `writeRaster()`
2. Sistema de coordenadas: `projection()` ou `@crs`
3. Resolução: `res()`
4. Extensão: `extent()` ou `@extent`
5. Domínio de valores: `@data`
6. Transformação de coordenadas: `projectRaster()`
7. Ler coordenadas dos pixels: `coordinates()`
8. Ler valores dos pixels: `values()`
9. Extracção de valores de pixels designados: `extract()`
10. Recorte e mosaicos: `crop()`, `merge()` e `mosaic()`
11. Declives, orientações, iluminação: `terrain()`, `hillShade()`
12. Filtros lineares (convolução) e não lineares: `focal()`

Em jeito de conclusão (vector)

Alguns funções (ou slots) definidos pelo package `sp`:

1. Importar/exportar: `readOGR()` e `writeOGR()`
2. Sistema de coordenadas: `proj4string()` ou `@proj4string`
3. Extensão: `bbox()` ou `@bbox`
4. Tabela de atributos: `@data`
5. Transformação de coordenadas: `spTransform()`
6. Coordenadas dos centroides de cada *feature*: `coordinates()`
7. Lista de geometrias das *features*: `@polygons` ou `@lines`
8. Lista de polígonos da *i*-ésima *feature*: `@polygons[[i]]@Polygons`
9. Lista de linhas da *i*-ésima *feature*: `@lines[[i]]@Lines`
10. Cruzamento de tabelas (*join*): `merge()`
11. Converter para raster: `rasterize()`; exemplo:

```
1 rasterize(x=icnf[6,],y=mde.etrs) # icnf[6,] é a AP Sintra-
    Cascais
```

Em jeito de conclusão (formatos de ficheiros)

1. Para saber quais formatos de ficheiros que podem ser lidos ou escritos com `readOGR()` e `writeOGR()`:

```
ogrDrivers()
```

2. Para saber quais formatos de ficheiros que podem ser lidos ou escritos com `raster()` e `writeRaster()`:

```
writeFormats()
```