

**Curso de Análise de Dados Geográficos com R**

**ISA, dezembro 2015**  
**SIGs com R: Dados *raster***

Manuel Campagnolo

Instituto Superior de Agronomia, Universidade de Lisboa

- 1 **Introdução**
- 2 **Dados geográficos matriciais em R**
- 3 **Interactividade gráfica em R**
- 4 **Alterar sistema de coordenadas**
- 5 **Combinar diversos sistemas de coordenadas**
- 6 **Amostragem espacial sobre uma imagem**
- 7 **Composição de imagens**
- 8 **Álgebra de imagens**
- 9 **Mosaicos de imagens**
- 10 **Análise do relevo**
- 11 **Filtros**

# Dados geográficos, ambiente R e aplicações SIG

## Processar informação geográfica em ambiente R

O objectivo do módulo é mostrar como se podem manipular dados geográficos em R. Em particular,

- 1 Ler, associar e alterar sistemas de coordenadas de um conjunto de dados geográficos;
- 2 Ler, criar e exportar conjuntos de dados geográficos de tipo:
  - 1 **matricial** (“raster”), e.g. formato **geotiff**;
  - 2 **vectorial**, e.g. formato **shapefile**.

Links sugeridos:

- 1 The Comprehensive R Archive Network’s task view “Analysis of Spatial Data”, by Roger Bivand
- 2 Edzer Pebesma and Roger Bivand, Classes and methods for spatial data in R, *R news*, 5/2 (2005) 9–14
- 3 Robert J. Hijmans, Introduction to the raster package, 2014

## Dados geográficos

As funções base do R permitem ler, visualizar e analisar dados. Neste módulo, o interesse é em **dados geográficos**, isto é, observações associadas a localizações geográficas.

Dados geográficos têm então duas componentes:

- A **localização das observações**, que é definida num certo **CRS**: sistema de coordenadas (geográficas ou cartográficas);
- Os **valores associados a essas observações** que podem ser representados por variáveis **contínuas** (e.g. temperatura, precipitação, elevação, índice de área foliar, área, etc) ou **categoriais** (e.g. nome de freguesia, tipo de ocupação do solo, tipo de solo, tipo de estrada, estado operacional de uma estação meteorológica, etc)

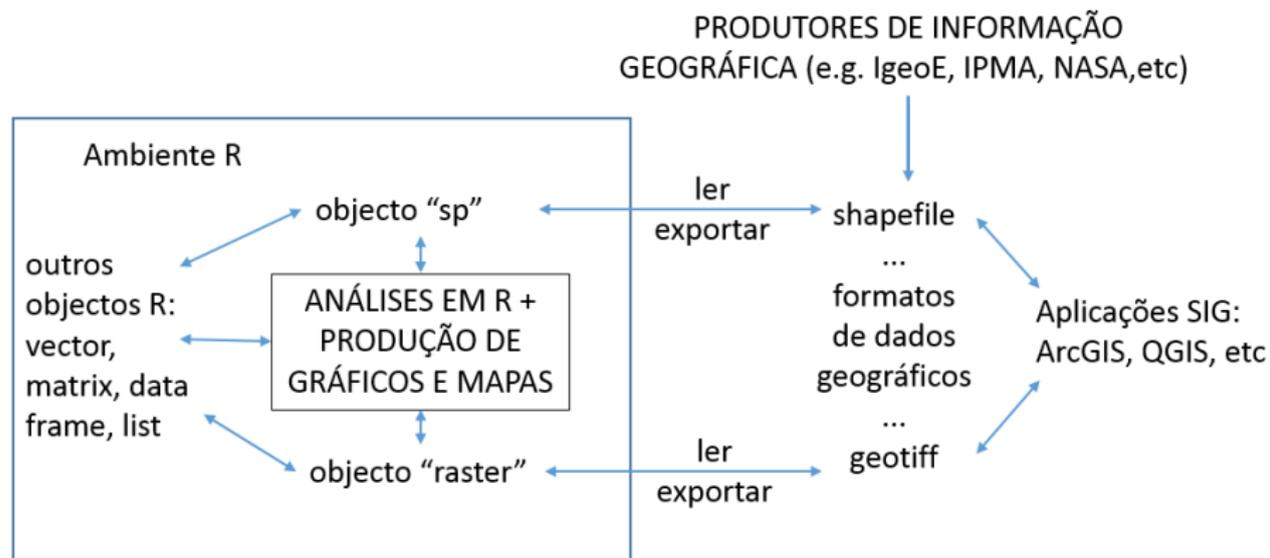
## Packages R para dados geográficos

Existem muitos *packages* disponíveis em R para dados geográficos (ver The Comprehensive R Archive Network's task view "Analysis of Spatial Data", by Roger Bivand para uma descrição do conjunto de *packages* disponíveis).

Essencialmente, os *packages* auxiliam o utilizador a:

- 1 **importar** para ambiente R dados geográficos e **exportar** dados geográficos que resultam da análise feita em R;
- 2 **analisar** dados geográficos em R.

# Relação entre dados geográficos, ambiente R e aplicações SIG



Por exemplo, uma imagem Landsat é descarregada do site `usgs.gov` em formato GeoTIFF. Pode ser lida em ambiente R através do comando `landsat<-raster(nomeFicheiro)` que devolve um objecto "raster". É legítimo fazer então `m<-as.matrix(landsat)`.

## Sistema de coordenadas (CRS): descrição proj.4

O sistema de coordenadas pode ser escrito de várias formas. Em R é comum usar-se a descrição `proj.4` como nos exemplos seguintes:

### 1 Coordenadas geográficas WGS84 :

```
"+proj=longlat +ellps=WGS84 +datum=WGS84"; ou  
"+init=epsg:4326";
```

### 2 Coordenadas projectadas ETRS-PT-TM06

```
"+ellps=GRS80 +towgs84=0,0,0 +proj=tmerc  
+lat_0=39d40'05.73''N+lon_0=08d07'59.19''W"; ou  
"+init=epsg:3763"
```

### 3 Coordenadas “militares” do IGeoE (Datum Lisboa)

```
"+proj=tmerc +towgs84=-304.046,-60.576,103.64  
+lat_0=39.666666666666666 +lon_0=1  
+k=1 +x_0=200000 +y_0=300000 +ellps=intl  
+pm=lisbon +units=m"; ou "+init=epsg:20790".
```

## Sistema de coordenadas (CRS): descrição proj.4

Uma boa fonte para descrições proj.4 e outras descrições de sistemas de coordenadas é o site <http://spatialreference.org/>.

Observação: quando o *datum* (elipsóide de referência e ponto de fixação) difere de um sistema de coordenadas para outro, é necessário fazer uma **transformação de datum**. O parâmetro `+towgs84` é usado para definir uma transformação do datum do sistema de coordenadas no datum WGS84.

Exemplos de transformação de Datum Lisboa em WGS84:

- 1 `+towgs84=-304.046,-60.576,103.64` com 3 parâmetros usado no exemplo anterior de sistema de coordenadas "militares";
- 2 uma alternativa é usar a transformação com 7 parâmetros `+towgs84=-283.1,-70.7,117.4,-1.16,0.06,-0.65,-4.1`.

Estruturas para dados geográficos matriciais  
(*raster*) em R: o package `raster`.

Transformações de sistemas de coordenadas  
em R

## Dados matriciais: leitura e escrita

A **leitura** de dados do tipo matricial pode ser feita com as funções:

- 1 `raster` do package `raster`; esta função converte dados matriciais (no ambiente R ou fora do ambiente R, i.e. num ficheiro) num objecto `RasterLayer`;
- 2 `readGDAL` do package `rgdal`; Esta função permite ler ficheiros de dados matriciais e tipicamente devolve um objecto `SpatialGridDataFrame`; `GDALInfo` devolve os parâmetros do conjunto de dados.

A **escrita** de dados pode ser feita com

- 1 `writeRaster` do package `raster`;
- 2 `writeGDAL` do package `rgdal`;

**Nota:** `raster` é tipicamente mais eficiente do que `readGDAL` para a leitura de conjuntos de dados de grande dimensão.

## Formatos para dados raster

Dados do tipo matricial representam o espaço de forma extensiva, e são em geral obtidos por detecção remota ou derivados digitalmente de informação espacialmente contínua.

Existem muitos formatos de dados matriciais que são produzidos por diferentes softwares e por diferentes produtores de informação. Os packages de R podem ler todos os formatos habituais.

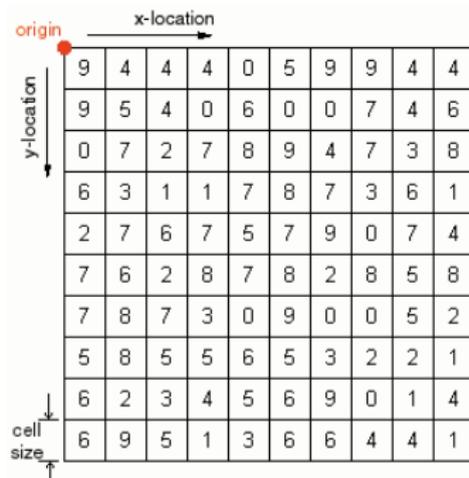
Muitos formatos suportam imagens com várias camadas como, por exemplo, imagens multiespectrais.

Correntemente o formato mais divulgado (que se tornou quase um standard) é o formato **Tag Image File Format** (TIFF) , com extensão `.tiff` ou `.tif`.

O formato TIFF admite um conjunto de informação adicional flexível (conhecidos por *tags*), consiste num ficheiro único com tamanho até 4Gb e é um formato do tipo **lossless**, isto é as técnicas de compressão usadas são totalmente reversíveis (não há perda de informação).

## Imagens, células e pixels

Todos os dados de tipo matricial (*raster*) têm uma estrutura semelhante: um **arranjo espacial regular** de valores. Cada elemento do arranjo (ou imagem) é designado por **célula** ou **pixel** e tem um valor numérico ou um não-valor (No data, NA, Null, ...)



**Figure:** Ilustração da estrutura de dados raster. No caso mais simples a georreferenciação é definida pelas coordenadas de um canto da imagem e pelo tamanho da célula (resolução espacial).

## Dados geográficos do tipo raster: o formato GeoTIFF

O formato GeoTIFF é o formato TIFF com **geolocation tags**. A georreferenciação é definida por:

- 1 **Raster space R** ou *image space*, usado para definir as coordenadas das células da imagem. O localização (0,0) corresponde ao canto superior esquerdo e está associada ao centro ou ao canto de uma célula de referência. A imagem tem um determinado número de linhas e de colunas.
- 2 **Model space M**, é usado para georreferenciar os dados; está associado a um sistema de coordenadas de referência (CRS);
- 3 **Transformação entre R e M**, que pode ser definida de várias formas, como apenas as coordenadas do canto (0,0), ou uma transformação linear da grelha, ou um conjunto de pontos de controlo, ou coeficientes racionais-polinomiais que permitem fazer a ortorectificação da imagem.

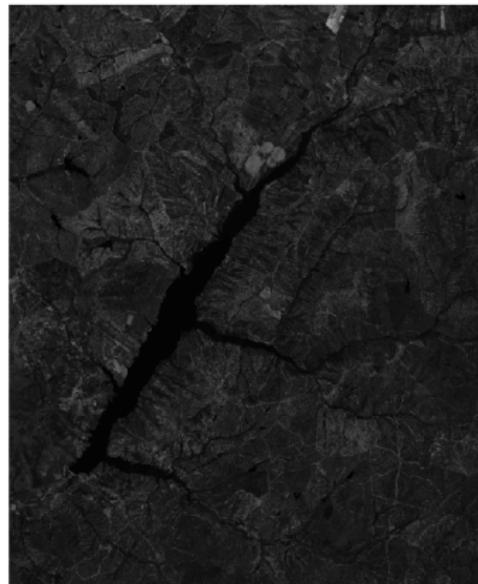
Designa-se **resolução** da imagem o tamanho de uma célula após a transformação. O domínio da imagem **data type** pode ser inteiro, *float* e tem um determinado tamanho (em bits).

## Dados matriciais: leitura e escrita com package raster:

Primeiro conjunto de dados: imagem pancromática Landsat 8.

- 1 Copiar `sessao_raster.zip` para a pasta de trabalho e abrir o script que contém os comandos R.
- 2 Executar os comandos:

```
1 setwd(paste0(getwd(), "\\dados_
  curso"))
2 # carregar package:
3 library(raster)
4 library(rgdal)
5 # ler ficheiro:
6 pan <- raster("landsat8pan.tif")
7 cores<-gray(seq(0,1,length.out
  =100))
8 plot(pan, xaxt="n", yaxt="n", box
  =FALSE, axes=FALSE, col=cores,
  horizontal=TRUE)
```



## Dados matriciais: leitura e escrita com package `raster`:

O objecto `pan` é um objecto R de classe `RasterLayer`:

```
1 pan # descreve o conjunto de dados
2 pan@crs # devolve o sistema de coordenadas em formato proj.4
   que é +proj=utm +zone=29 +datum=WGS84 +units=m +no_defs +
   ellps=WGS84 +towgs84=0,0,0
3 # exportar para ficheiro GeoTIFF
4 writeRaster(r, file="out.tif", format="GTiff", overwrite=TRUE)
```

Nota. O package `raster` define várias classes de objectos (`RasterLayer`, `Extent`, ...). Cada classe contém *slots* que servem a armazenar informação da classe. O acesso a *slots* faz-se com `@` (e.g. `pan@crs`) ou com a função `slot`.

Nota: as opções de plot para objectos `raster` são listadas com `args(raster:::rasterImagePlot)`.

## Análise de dados com package `raster`: valores das células

A imagem obtida com `plot(pan)` usa uma paleta de cores por omissão. Como é habitual com imagens pancromáticas, foram usados tons de cinzento. Foi também eliminada a caixa e os eixos em volta da figura. Essas modificações foram obtidas com os seguintes comandos:

```
1 # criar vector de 100 tons de cinzento entre preto (0) e branco
   (1)
2 cores<-gray(seq(0,1,length.out=100))
3 plot(pan,col=cores)
4 # tirar caixa e colocar a legenda na horizontal
5 plot(pan, xaxt="n", yaxt="n", box=FALSE, axes=FALSE, col=cores
      ,horizontal=TRUE)
```

A imagem está escura dado que `plot` transforma linearmente os valores das células num intervalo  $[0,1]$ , ou seja, transforma o valor mínimo em 0, e o valor máximo em 1, e todos os valores intermédios através da recta correspondente.

## Alteração de contraste em imagens

Um conjunto de dados do tipo raster tem **valores das células**  $\{z_1, \dots, z_n\}$  com domínio  $[z_{\min}, z_{\max}]$ . O  $j$ -ésimo valor é mostrado no monitor com uma intensidade  $I_j$  entre  $I_{\min}$  (por exemplo, 0) e  $I_{\max}$  (por exemplo 1). Em geral,  $I_{\min}$  corresponde a **preto** e  $I_{\max}$  a **branco**.

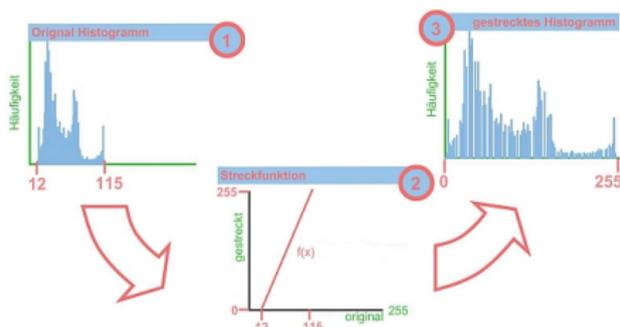
Sem alteração de contraste, a intensidade  $I$  é uma função linear de  $z$  sobre  $[z_{\min}, z_{\max}]$  que não altera o histograma; por isso a imagem pode ficar muito escura ou muito clara. A **alteração do contraste** é definida por uma função crescente, não linear,  $I = f(z)$ .

No caso mais simples

$[z_{\text{new min}}, z_{\text{new max}}]$  é o suporte do histograma e  $f$  é linear por partes com

$$f(z) = I_{\min}, \text{ se } z < z_{\text{new min}},$$

$$f(z) = I_{\max}, \text{ se } z > z_{\text{new max}}.$$



## Análise de dados com `package raster`: valores das células

Os valores das células de um objecto `RasterLayer` podem ser lidos para o ambiente R com a função `values`:

```
1 v<-values(pan) # vector numérico
```

Os valores mínimo e máximo dos pixels podem ser obtidos com

```
1 range(v)
```

Nota: os slots `@data@min` e `@data@max` devolvem esses mesmos valores:

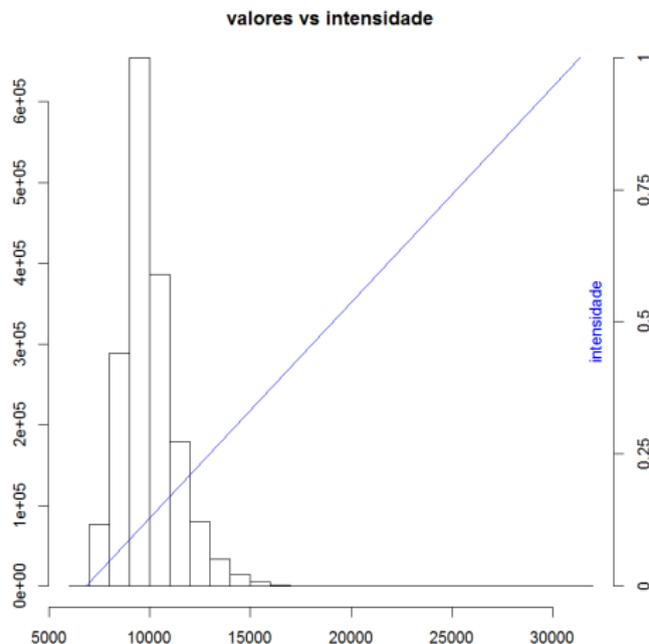
```
1 pan@data@min  
2 pan@data@max
```

Nota: a leitura de cdg matriciais com `raster` não coloca a totalidade dos dados na sessão R por uma questão de eficiência; a função `values` lê os valores no ficheiro "landsat8pan.tif" e guarda-os em memória na sessão, o que pode ser demasiado pesado.

## Análise de dados com package `raster`: valores das células

Em seguida é desenhado o histograma para perceber como é que as intensidades na imagem são calculadas:

```
1 out<- hist(v, main="valores vs
  intensidade")
2 fqs<-out$counts # frequências
  absolutas das classes
3 # representar a transformação
  linear de valores dos
  pixels em intensidade
4 lines(x=range(v), y=range(fqs),
  col="blue")
5 # representar os valores de
  intensidade
6 axis(4, at=seq(min(fqs), max(fqs)
  , length.out=5), labels=seq
  (0, 1, length.out=5))
7 mtext("intensidade", side=4,
  line=-1.5, col="blue")
```



Nota: `hist(pan)` uma apenas uma amostra dos valores.

## Análise de dados com package raster: alteração de contraste

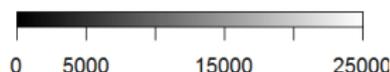
Uma forma de aumentar o contraste é saturar a imagem (para valores baixos, i.e. escuros) ou valores elevados. Neste caso, a observação do histograma sugere saturar a imagem para valores elevados.

Exemplo: aumentar o contraste fixando o valor 25000 para a saturação da imagem:

```
1 plot(pan, zlim=c(0,25000), xaxt="n"  
      , yaxt="n", box=FALSE, axes=FALSE  
      , col=cores, horizontal=TRUE)
```

Em alternativa pode saturar-se pelo quantil 0.999:

```
1 plot(pan, zlim=c(0,quantile(v,.999)  
      ), xaxt="n", yaxt="n", box=FALSE,  
      axes=FALSE, col=cores, legend=  
      FALSE)
```



## Análise de dados com package `raster`: coordenadas $x, y$

Muitas funções permitem extrair informação da imagem. Em particular é possível obter as coordenadas dos centros de todos os pixels da imagem e a dimensão desses pixels.

`coordinates` devolve as coordenadas dos centros das células; mais precisamente devolve uma matrix com duas colunas que correspondem a  $x$  e a  $y$ ;

`res` devolve a resolução nas duas direcções;

`extent` devolve um objecto com a extensão da imagem;

```
1 head(coordinates(pan))  
2 res(pan)  
3 extent(pan)
```

Em analogia a `zlim` pode usar-se `xlim` e `ylim` para restringir o a imagem produzida com `plot` a uma região mais pequena:

```
1 plot(pan, xlim=c(570000,575000), ylim=c(4315000,4325000), zlim=  
      c(0,25000), xaxt="n", yaxt="n", box=FALSE, axes=FALSE, col=  
      cores, legend=FALSE)
```

## Dados matriciais: interface com “rato” e `extent`

O ambiente R é muito limitado para manipular interactivamente dados geográficos. No entanto existem funções que permitem alguma interactividade tais como `zoom`, `click`, `locator` ou `drawExtent`.

A função `drawExtent` permite usar o rato sobre a imagem para definir um objecto de classe `extent`:

```
1 plot(pan, xaxt="n", yaxt="n", box=FALSE, axes=FALSE, legend=
  FALSE, col=cores)
2 box <- drawExtent() # clicar em 2 cantos
3 # definir a extensão da imagem usando box
4 plot(pan, ext=box, xaxt="n", yaxt="n", box=FALSE, axes=FALSE,
  legend=FALSE, col=cores)
```

Pode manipular-se directamente o objecto da classe `extent` para alterar a extensão da figura. No exemplo abaixo, usa-se apenas a metade central de `box`:

```
1 plot(pan, ext=box/2, xaxt="n", yaxt="n", box=FALSE, axes=FALSE,
  legend=FALSE, col=cores)
```

## Dados matriciais: interface com “rato” e `extent`

A função `zoom` permite fazer um “zoom” numa região rectangular:

```
1 plot(pan, xaxt="n", yaxt="n", box=FALSE, axes=FALSE, legend=
  FALSE, col=cores)
2 zoom(pan, new=FALSE, xaxt="n", yaxt="n", box=FALSE, axes=FALSE,
  legend=FALSE, col=cores) # clicar em 2 cantos
```

Pode aceder-se aos valores das células interactivamente com `click`:

```
1 click(pan) # devolve valor de pixels; parar com <ESC>
```

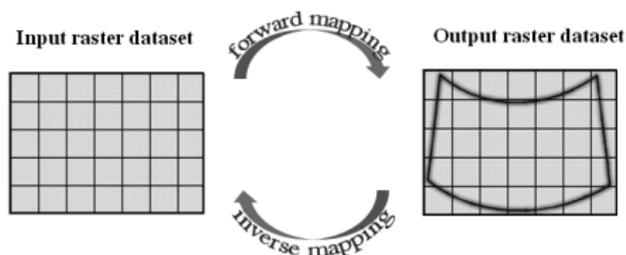
Pode digitalizar-se um polígono com `locator`

```
1 pts <- locator() # devolve uma lista de coords; parar com <ESC>
2 polygon(cbind(pts$x,pts$y),border="red") # adiciona à imagem o
  polígono formado pelos pontos digitalizados
```

Pode usar-se `extent` ou algum objecto que possa ser interpretado como tal para `recortar` `pan` e criar um novo `RasterLayer`:

```
1 pan.crop <- crop(x=pan,y=pts) # a nova extensão é dada por pts
2 plot(pan.crop, xaxt="n", yaxt="n", box=FALSE, axes=FALSE,
  legend=FALSE, col=cores)
```

## Análise de dados com `package raster`: alterar sistema de coordenadas



A re-projecção de um cdg matricial, com um dado CRS, num novo cdg matricial com outro CRS envolve três aspectos:

- 1 transformação de coordenadas  $f(x, y) = (u, v)$ ;
- 2 características da grelha de output (extensão, resolução);
- 3 critério de re-amostragem.

## Análise de dados com package `raster`: alterar sistema de coordenadas

O package `raster` contém a função `projectRaster` que permite **re-projectar** os dados, definir a nova **resolução** e **extensão**, e escolher o critério de **re-amostragem**.

`projectRaster` usa os sistemas de coordenadas (CRS) do input e do output. Como visto atrás, a forma mais simples de definir o CRS é através de uma descrição `proj.4`.

A descrição `proj.4` contém em particular:

- 1 A definição da projecção cartográfica, e.g. `+proj=tmerc`;
- 2 A definição do datum, e.g. `+datum=WGS84`;
- 3 A descrição da transformação de datum relativamente ao datum WGS84, e.g.

```
+towgs84=-304.046,-60.576,103.64,0,0,0,0,  
+towgs84=-283.1,-70.7,117.4,-1.16,0.06,-0.65,-4.1  
ou +nadgrids=ptLX_e89.gsb.
```

## Análise de dados com package `raster`: alterar sistema de coordenadas

Exemplo 1: re-projectar indicando apenas o CRS do output; neste caso, a **extensão** e **resolução** são iguais às do input; o critério de re-amostragem é **bi-linear** por omissão.

```
1 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" #
    Coordenadas 'militares'
2 m.igeoe <- projectRaster(pan, crs=igeoe)
```

Exemplo 2: criar primeiro um objecto `RasterLayer` vazio com a **extensão** e **resolução** desejados, e em seguida fazer a re-projecção do input com `projectRaster`; no exemplo é também escolhido o critério de **re-amostragem** com o argumento `method`:

```
1 vazio <- raster(xmn=195000, xmx=205000, ymn=230000, ymx=245000,
    resolution=100, crs=igeoe)
2 m2 <- projectRaster(from=pan, to=vazio, method="ngb")
3 m3 <- projectRaster(from=pan, to=vazio, method="bilinear")
```

## Análise de dados com package `raster`: alterar sistema de coordenadas

Ilustração: comparar input `pan` e output `m.igeoe` da re-projecção.

Representar dados originais em coordenadas UTM, zona 29:

```
1 par(mfrow=c(2,1))
2 plot(pan, xaxt="n", yaxt="n", box=FALSE, axes=FALSE, legend=
  FALSE, col=cores, zlim=c(0,25000))
3 xy<-as.vector(pan@extent)
4 # xpd para poder escrever fora da imagem
5 text(xy[c(1,2,1,1)],xy[c(3,3,3,4)],round(xy),pos=c(1,1,2,2),xpd
  =TRUE)
```

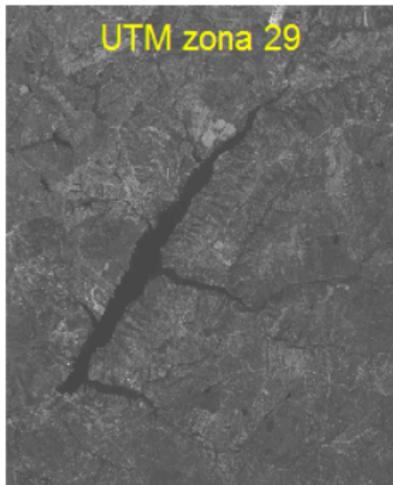
Representar a imagem transformada para coordenadas militares, com uma transformação de datum definida por sete parâmetros (`igeoe`):

```
1 plot(m.igeoe, xaxt="n", yaxt="n", box=FALSE, axes=FALSE, legend
  =FALSE, col=cores, zlim=c(0,25000))
2 xy<-as.vector(m@extent) # extensão xmin, xmax, ymin, ymax
3 text(xy[c(1,2,1,1)],xy[c(3,3,3,4)],round(xy),pos=c(1,1,2,2),xpd
  =TRUE)
```

# Análise de dados com package `raster`: sistemas de coordenadas

4340542

UTM zona 29



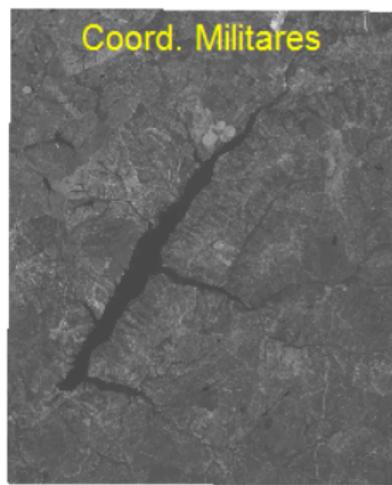
4318778

568898

586702

249358

Coord. Militares



227263

193769

211934

## Análise de dados com `package raster`: alterar sistema de coordenadas

A transformação de datum (que é sempre relativa a WGS84) pode ser feita de duas formas:

- 1 através de um conjunto de parâmetros (transformação 3D conhecida por Helmert ou Bursa-Wolf), e.g.  
`+towgs84=-283.1,-70.7,117.4,-1.16,0.06,-0.65,-4.1;`
- 2 através de grelhas (conhecidas por "NAD grids") , e.g.  
`+nadgrids=ptLX_e89.gsb.`

Neste caso, é necessário:

- 1 colocar o ficheiro com a informação necessária (e.g. "ptLX\_e89.gsb") na pasta indicada por `system.file("proj", package = "rgdal")`; nota: é necessário fazer o *restart* do R para ter efeito;
- 2 indicar na definição do CRS que a transformação de datum é do tipo "nadgrid" com o parâmetro `+nadgrids=ptLX_e89.gsb`;
- 3 aplicar a re-projecção da forma habitual.

## Análise de dados com package `raster`: projecção com grelhas

Exemplo: Re-projectar `cdg_pan` pelo método das grelhas.

Definir CRS com parâmetro `+nadgrids`, e re-projectar:

```
1 igeoe.grid <- CRS("+proj=tmerc +lat_0=39.66666666666666 +  
    nadgrids=ptLX_e89.gsb +lon_0=1 +k=1 +x_0=200000 +y_0=300000  
    +ellps=intl +pm=lisbon +units=m")  
2 m.grid <- projectRaster(pan, crs=igeoe.grid)
```

Representar o resultado:

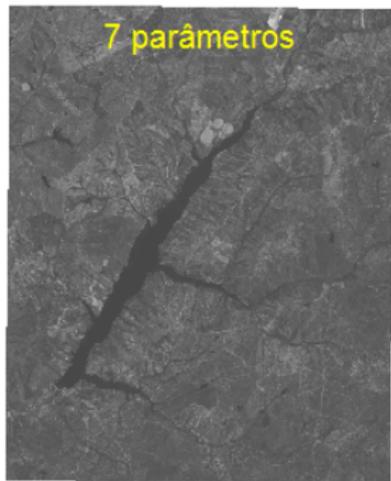
```
1 plot(m.grid , xaxt="n" , yaxt="n" , box=FALSE , axes=FALSE , legend=  
    FALSE , col=cores , zlim=c(0,25000))  
2 xy<-as.vector(m.grid@extent)  
3 text(xy[c(1,2,1,1)] , xy[c(3,3,3,4)] , round(xy) , pos=c(1,1,2,2) , xpd  
    =TRUE)
```

## Análise de dados com package `raster`: projecção com grelhas

Comparação do cdg `m.igeoe`, obtido através do CRS `igeoe`, que usa 7 parâmetros para a transformação de datum, com o cdg `m.grid` obtido usando o método das grelhas e que é em geral mais preciso.

Para este caso as diferenças são inferiores a 1m mas podem ser maiores em outras zonas de Portugal.

249356

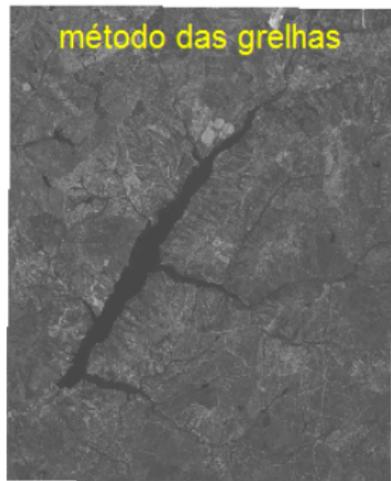


227261

193769

211934

249356



227261

193769

211934

## Exercício: utilização de `projectRaster` para determinar a localização de um ponto num outro CRS

Usando o objecto `pan`, em coordenadas UTM zona 29, que representa uma região em redor da albufeira da Barragem de Montargil, determine:

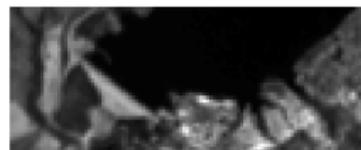
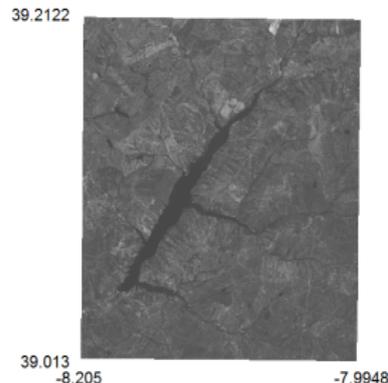
- 1 a gama de latitudes e longitudes que corresponde a essa imagem;
- 2 a localização da parede da barragem em lat/long.

Sugestão: Use `projectRaster` e o CRS de coordenadas geográficas lat/long para obter as novas coordenadas. Para localizar bem a parede da barragem use `zoom` e para identificar use `locator`.

*Solução:* aproximadamente lat=39.05347 N e long=8.176397 W.

## Solução do exercício

```
1 # projectar raster em WGS84
2 wgs84<-"+proj=longlat +ellps=WGS84 +
  datum=WGS84"
3 w <- projectRaster(pan, crs=wgs84)
4 plot(w, xaxt="n", yaxt="n", box=FALSE,
  axes=FALSE, legend=FALSE, col=cores
  , zlim=c(0,25000))
5 # Determinar a extensão
6 xy<-as.vector(w@extent)
7 text(xy[c(1,2,1,1)],xy[c(3,3,3,4)],
  round(xy,4), pos=c(1,1,2,2), xpd=TRUE
  )
8 # Fazer zoom sobre parede da barragem
9 zoom(w, new=FALSE, xaxt="n", yaxt="n",
  box=FALSE, axes=FALSE, legend=FALSE
  , col=cores) # clicar sobre 2
  pontos
10 # Usar locator para obter coordenadas
  lat/long da parede da barragem
11 locator() # parar com <ESC>
```



## Exercício

Neste exercício pretende-se comparar os valores de elevação com os valores de índice de vegetação “NDVI” para a área protegida Sintra-Cascais. Os dados de input estão em três sistemas de coordenadas distintos:

- 1 Dados NDVI derivados de uma imagem Landsat 8 de Março de 2014 com CRS UTM zona 29 e resolução de 30m – ficheiro “ndviSintra.tif”;
- 2 Modelo digital de elevações (MDE) SRTM com CRS WGS84 e resolução de 3-arc segundos – ficheiro “n38\_w010\_3arc\_v2.tif”;
- 3 Limite da área protegida em CRS ETRS-PT-TM06, extraído de cartas disponíveis na página do ICNF – ficheiro “limite.AP.SintraCascais.ETRS.txt”. O ficheiro tem duas colunas que correspondem à coordenada x e à coordenada y no CRS ETRS-PT-TM06 de pontos que definem o contorno da área protegida.

## Análise de dados matriciais: exercício AP Sintra-Cascais

Ler ficheiro com as coordenadas (ETRS) do limite da área protegida para matriz:

```
1 limite.ap <- as.matrix(read.table(file="limite.AP.SintraCascais  
   .ETRS.txt",header=FALSE))
```

Ler ficheiro com dados de elevação (MDE):

```
1 mde <- raster("n38_w010_3arc_v2.tif")
```

Converter MDE para ETRS e recortar pelo limite da AP:

```
1 etrs <- "+proj=tmerc +lat_0=39.6682583 +lon_0=-8.1331083 +k=1 +  
   x_0=0 +y_0=0 +ellps=GRS80 +units=m"  
2 mde.etrns <- projectRaster(mde,crs=etrns)  
3 mde.etrns <- crop(mde.etrns,limite.ap)
```

## Análise de dados matriciais: exercício AP Sintra-Cascais

Ler imagem de índice de vegetação NDVI:

```
ndvi<-raster("ndviSintra.tif")
```

Re-projectar NDVI para ETRS alinhando a imagem com `mde.etr`s. O alinhamento é feito escolhendo como extensão e como resolução do novo `RasterLayer` a extensão e a resolução de `mde.etr`s:

```
1 ndvi<-raster("ndviSintra.tif")
2 vazio<-raster(ext=mde.etr@extent, crs=mde.etr@crs, resolution=
  res(mde.etr))
3 ndvi.etr<-projectRaster(from=ndvi, to=vazio)
```

Nota: para re-projectar um cdg sobre a quadrícula de outro cdg matricial, não é necessário conhecer a extensão, resolução e CRS do cdg original: basta extrair essa informação com `@extent`, `@crs`, e com a função `res`.

## Análise de dados matriciais: exercício AP Sintra-Cascais

Para obter uma **amostra** de pares de valores de NDVI e de elevação pode usar-se a função `spsample` do package `sp` que permite fazer uma amostragem espacial na extensão de `spsample`.

Alternativamente, pode realizar-se uma escolha aleatória de 2000 pontos na extensão de `limite.ap` usando a função `sample`.

```
1 minx<-min(limite.ap[,1]); maxx<-max(limite.ap[,1])
2 miny<-min(limite.ap[,2]); maxy<-max(limite.ap[,2])
3 amostra2D<-cbind(sample(x=minx:maxx, size=2000), sample(x=miny:
  maxy, size=2000))
```

Note-se que os 2000 pontos escolhidos por `spsample` ou pelo exemplo acima não estão garantidamente no interior do polígono `limite.ap` dado que apenas é garantido estarem no menor rectângulo que contém `limite.ap`. Será então necessário seleccionar o subconjunto dos pontos escolhidos que estão no polígono.

## Análise de dados matriciais: exercício AP Sintra-Cascais

A selecção dos pontos de `amostra2D` que estão no interior do polígono `limite.ap` pode ser realizado com a função `points.in.polygon` do package `sp`. O valor devolvido pela função pode ser 0, 1, 2, ou 3 para cada ponto testado. É 1 quando o ponto está no interior do polígono.

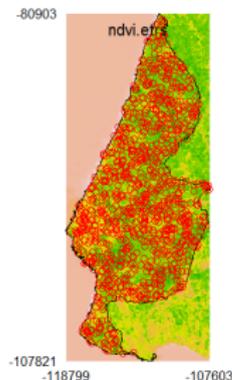
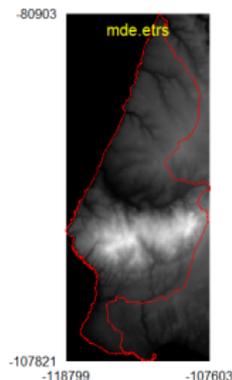
```
1 library(sp)
2 # vector TRUE/FALSE do tamanho de amostra:
3 dentro<-point.in.polygon(point.x=amostra2D[,1], point.y=
      amostra2D[,2], pol.x=limite.ap[,1], pol.y = limite.ap[,2])==1
4 # selecção dos pontos dentro de limite.ap:
5 amostra2D.ap<-amostra2D[dentro,]
```

Finalmente é necessário **extrair** os valores dos raster `mde.etr`s e `ndvi.etr`s nos pontos da amostra com a função `extract`:

```
1 y<-extract(ndvi.etr, amostra2D.ap)
2 x<-extract(mde.etr, amostra2D.ap)
```

## Análise de dados matriciais: exercício AP Sintra-Cascais

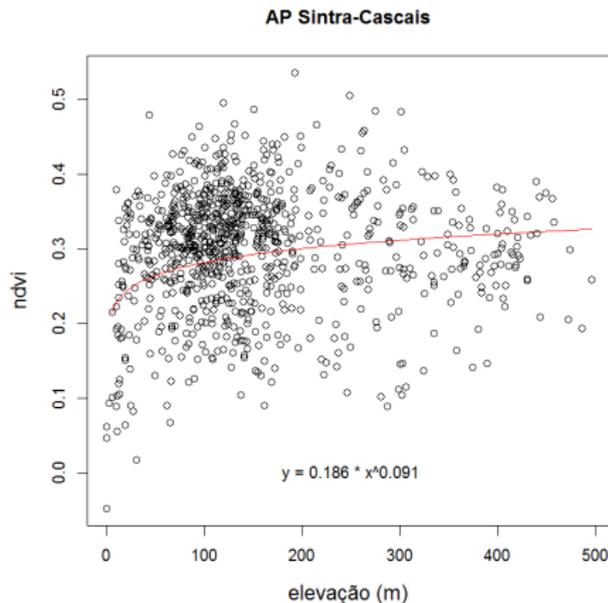
```
1 par(mfrow=c(2,1),mar=rep(2,4))
2 plot(mde.etr, xaxt="n", yaxt="n", box=
  FALSE, axes=FALSE, col=cores,legend=
  FALSE)
3 polygon(limite.ap, border="red")
4 xy<-as.vector(mde.etr@extent)
5 text(xy[c(1,2,1,1)],xy[c(3,3,3,4)],round(
  xy,0),pos=c(1,1,2,2),xpd=TRUE,cex=0.8)
6 text(x=mean(xy[1:2]),y=xy[4],"mde.etr",
  col="yellow",pos=1)
7
8 plot(ndvi.etr, xaxt="n", yaxt="n", box=
  FALSE, axes=FALSE, legend=FALSE)
9 polygon(limite.ap, border="black")
0 xy<-as.vector(ndvi.etr@extent)
1 text(xy[c(1,2,1,1)],xy[c(3,3,3,4)],round(
  xy,0),pos=c(1,1,2,2),xpd=TRUE,cex=0.8)
2 text(x=mean(xy[1:2]),y=xy[4],"ndvi.etr",
  col="black",pos=1)
3 points(amostra2D.ap,col="red")
```



## Análise de dados matriciais: exercício AP Sintra-Cascais

Pode agora construir-se um gráfico e ajustar uma curva  $y = ax^b$  aos dados:

```
1 plot(y~x, xlab="elevação (m)",
      ylab="ndvi", main="AP Sintra
      -Cascais", cex.lab=1.3)
2 xx<-log(x[x>0]) # transformação
  logarítmica
3 yy<-log(y[x>0])
4 ajust<-lm(yy~xx) # ajustar
  regressão
5 curve(exp(ajust$coef[1])*x^
      ajust$coef[2], add=TRUE, col=
      "red")
6 text(mean(range(x, na.rm=TRUE))
      ,0, paste("y = ", round(exp
      (ajust$coef[1]), 3), " * x^"
      ,round(ajust$coef[2], 3), sep
      =""))
```



## Composição de imagens com o package `raster`

## Análise de dados com package `raster`: composição de imagens

A classe `raster` contém objectos `RasterBrick` e `RasterStack` que suportam **múltiplas imagens**. Cada imagem deve ter a mesma **extensão** e **resolução**. `RasterStack` é mais flexível (por exemplo, pode ser formado lendo vários ficheiros) e `RasterBrick` é uma estrutura de dados mais rígida mas que permite um processamento mais eficiente.

No próximo exemplo usa-se a função `stack` para ler várias bandas Landsat 7 da mesma região (cada banda é um ficheiro GeoTIFF). Depois, converte-se o resultado para um `RasterBrick`.

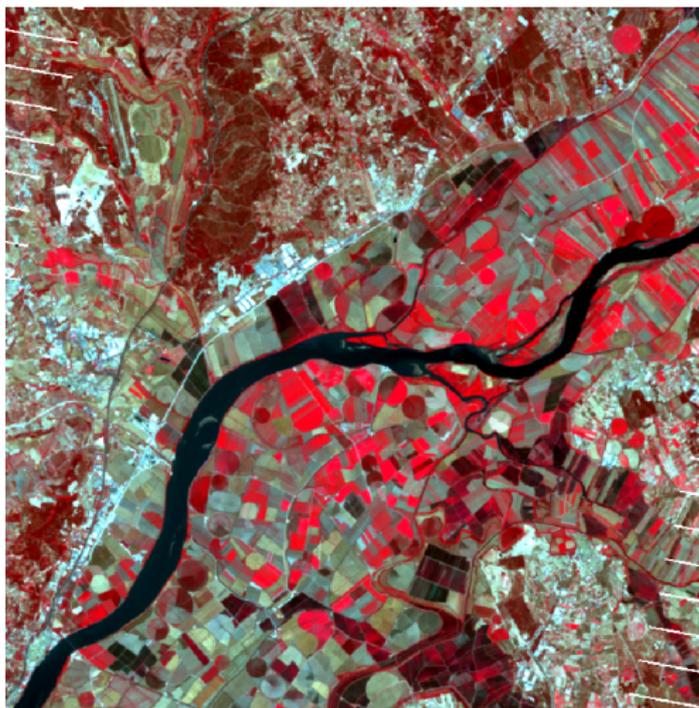
```
1 fichs <- list.files(pattern="banda[1-7]")
2 s <- stack(as.list(fichs))
3 s # devolve sumário dos dados
4 b <- brick(s)
5 b # idem
6 nlayers(b) # devolve número de camadas em b
7 box <- extent(c(500000, 520000, 4310000, 4330000))
```

## Análise de dados com package `raster`: Composição Landsat RGB=432, Ribatejo

A função `plotRGB` permite construir combinações coloridas de imagens. Vamos exemplificar com uma combinação RGB=432.

```
plotRGB(b, r=4, g=3, b=2,  
        stretch="lin", ext=  
        box)
```

Neste caso, há uma alteração linear do contraste e a figura é realizada apenas na extensão definida pela `box` definida atrás.



É possível, em ambiente R, ter acesso a várias fontes de dados matriciais, e em particular, a imagens Google Maps. Existem vários packages que permitem o acesso e "download" dessas imagens, que ficam disponíveis na sessão R e podem ser manipuladas como qualquer outro conjunto de dados do R. Em particular, pode aceder-se a cada um dos canais RGB dessas imagens.

Um dos packages com essa funcionalidade é o package `dismo`.

## Análise de dados com package `raster`: dados Google maps

Vamos escolher uma pequena zona (por forma a ter melhor detalhe) para a qual se pretende ter as imagens de muito alta resolução Google Maps, na parte central do stack `b` (no Ribatejo) usado anteriormente.

Em primeiro vamos verificar que `b` tem coordenadas UTM, zona 29:

```
1 b@crs
```

Vamos extrair uma pequena região da imagem Landsat que poderia também ter sido obtido com `drawExtent()`:

```
1 e<-extent(as.vector(c( 513316, 514509, 4317568 ,4318417)))  
2 landsat<-crop(b,e)
```

A imagem Google é extraída com coordenadas lat/long:

```
1 wgs84<-"+proj=longlat +ellps=WGS84 +datum=WGS84"  
2 aux<-projectRaster(landsat , crs=wgs84)  
3 # o slot @extent devolve um objecto de classe extent  
4 aux@extent # coordenadas lat/long do objecto landsat  
5 zona.central<-aux@extent/2 # coordenadas lat/long da parte  
   central da extensão dada por aux@extent
```

## Análise de dados com package `raster`: dados Google maps

Sabendo a extensão da zona em que queremos extrair a imagem Google, pode usar-se então a função `gmap` do package `dismo` para aceder a essa imagem. O argumento `rgb` permite extrair a composição colorida (`rgb=FALSE` devolve um `RasterLayer`) ou as três bandas (`rgb=TRUE` devolve um `RasterBrick`). O argumento `scale` pode ser 1 ou 2 (resolução duas vezes melhor).

```
1 install.packages("dismo")
2 library(dismo)
3 gm.wgs<-gmap(zona.central, type="satellite", lonlat=TRUE, rgb=TRUE
  , scale=2)
```

Verificar que o CRS de `gm.wgs` é WGS84, e que as bandas se designam por "red", "green", "blue".

```
1 gm.wgs
```

## Análise de dados com package `raster`: dados Google maps

Em seguida vamos verificar visualmente que as imagens `gm.wgs` (CRS WGS84) e `landsat` (CRS UTM, zona 29) se sobrepõem correctamente.

- 1 Re-projectar os dados num novo CRS: ETRS-PT-TM06.

```
1 etrs<-"+proj=tmerc +lat_0=39.6682583 +lon_0=-8.1331083 +k=1  
  +x_0=0 +y_0=0 +ellps=GRS80 +units=m"  
2 gm. etrs<-projectRaster (gm.wgs , crs=etrs )  
3 landsat. etrs<-projectRaster (landsat , crs=etrs )
```

- 2 Em seguida, construir a sobreposição usando parâmetro `alpha` que varia entre 0, para transparente, e 255, para opaco. A composição `landsat` é `RGB=432`

Qual é a resolução (em metros) da imagem Google descarregada?

## Análise de dados com package `raster`: Sobreposição de uma composição Landsat RGB=432 sobre um mapa de alta resolução (Google maps) no Ribatejo

```
1 par(mfrow=c(1,1))
2 # composição Google
  maps em cor
  verdadeira
3 plotRGB(gm. etrs , r=3,g
  =2,b=1,stretch="
  lin")
4 # composição Landsat
  RGB=432
5 # alpha define
  transparência
6 plotRGB(landsat. etrs , r
  =4,g=3,b=2,stretch
  ="lin",ext=gm.
  etrs@extent , alpha
  =100,add=TRUE)
```



## Análise de dados com package `raster`: álgebra de imagens

Os valores de um objecto `raster` podem ser acedidos com a função `values` que devolve um vector numérico. Mas é normalmente mais simples usar directamente o objecto `raster`. No exemplo abaixo calcula-se o NDVI a partir das bandas 3 e 4 de `RasterBrick` `b`.

```
1 ndvi <- (b$banda4 - b$banda3)/(b$banda4 + b$banda3)
```

Os valores são manipulados como numa matriz, e.g. determinar a proporção de pixels de `ndvi` que têm valor `NA`, ou alterar valores:

```
1 ncell(ndvi[is.na(ndvi)])/ncell(ndvi) #devolve a proporção de  
   pixels com NA; pode usar-se length em vez de ncell  
2 ndvi[ndvi<0] <- 0 # substituição de valor
```

Existem funções de mais alto nível tais como `reclassify`, `subs` ou `cut` que permitem alterar os valores das células usando uma "lookup table". Abaixo reclassifica-se `ndvi` de acordo com a transformação  $] - 1, 0] \rightarrow 1$ ,  $]0, .5] \rightarrow 2$  e  $] .5, 1] \rightarrow 3$ :

```
1 tabela <- cbind(c(-1, .2, .5), c(.2, .5, 1), 1:3)  
2 ndvi.c <- reclassify(ndvi, rcl=tabela, right=TRUE)
```

## Análise de dados com package `raster`: álgebra de imagens

Um conjunto de funções aplica-se a `RasterBrick` e `RasterStack` célula a célula devolvendo um novo `raster`

```
1 min(b) # RasterLayer dos mínimos nas 6 bandas  
2 range(b) # devolve RasterBrick com min e max
```

Para obter uma estatística para cada banda usa-se `cellStats`:

```
1 cellStats(b, "mean") # devolve um vector com 6 componentes
```

Operações numéricas ou lógicas sobre um ou mais objectos `raster` podem ser realizadas com as funções `calc` e `overlay` respectivamente; `fun` tem que aceitar o mesmo número de argumentos que o número de camadas de `RasterBrick` ou `RasterStack`:

```
1 median(b) #dá erro  
2 b.mediana<-overlay(b, fun=median) # lento , mas funciona
```

## Análise de dados com package `raster`: mosaico de imagens

Podemos criar um mosaico a partir de um conjunto de imagens alinhadas, com a mesma resolução e CRS. Existem para tal duas funções no package `raster`:

- 1 a função `merge` que usa a primeira imagem do conjunto quando há sobreposição;
- 2 a função `mosaic` que é mais flexível, e que permite usar qualquer função das várias imagens quando há sobreposição.

A função tem um argumento `fun` que é aplicado ao conjunto de valores de células com a mesma localização. A função indicada por `fun` tem que ter obrigatoriamente um argumento `na.rm` para poder ser aplicada a células com valor NA. Por exemplo,

```
fun=min,
```

```
fun=mean,
```

```
fun=function(x, ...) quantile(x, .95).
```

## Análise de dados com package `raster`: mosaico de imagens

Vamos usar dados de altimetria SRTM disponíveis on-line. Primeiro vamos criar uma string com a URL onde estão os dados:

```
urlzip<- "http://dds.cr.usgs.gov/srtm/version2_1/SRTM3/Eurasia/N37W008.hgt.zip" # cobre SE de Portugal
```

Depois, usa-se a função do R `download.file` para descarregar o ficheiro para a pasta de trabalho:

```
download.file(url=urlzip, destfile="N37W008.hgt.zip", mode="wb")
```

Como o ficheiro está comprimido, usa-se `unzip` para descomprimir na pasta de trabalho:

```
unzip(zipfile="N37W008.hgt.zip")
```

Finalmente, usa-se `raster`, que lê em particular o formato "hgt":

```
srtm8<-raster("N37W008.hgt")
```

Fazer o mesmo para descarregar o ficheiro `N37W009`, e obter `srtm9`.

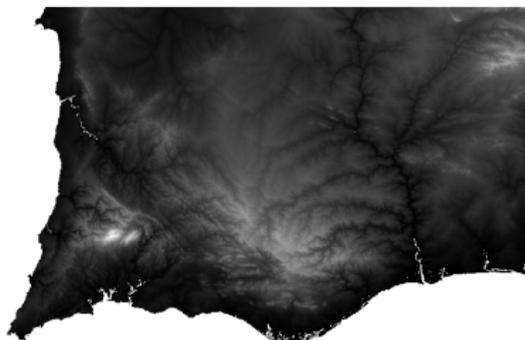
## Análise de dados com package `raster`: mosaico de imagens

Para os dados do exemplo que está a ser considerado, não há sobreposição das imagens e por isso o mosaico pode ser simplesmente obtido com a função `merge`:

```
1 srtm <- merge(srtm8 , srtm9)  
2 srtm[srtm<=0]<-NA # para não representar elevações negativas
```

Representar o resultado (`cores` são tons de cinzento):

```
1 plot(srtm , xaxt="n" , yaxt="n" , box=FALSE , axes=FALSE , legend=  
      FALSE , col=cores)
```



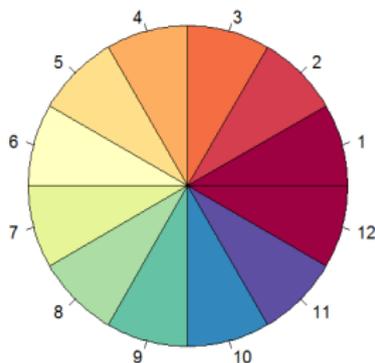
## Análise de dados com package `raster`: cores

Para tornar a figura mais apelativa, podemos usar uma escala de cores apropriada para a altimetria. O package `RColorBrewer` permite definir algumas escalas de cores interessantes:

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

Para ver em mais detalhe uma paleta particular com a função `pie`:

```
1 N<-10; pal<-"Spectral"; pie(rep(1,N), col=brewer.pal(n=N,pal))
```



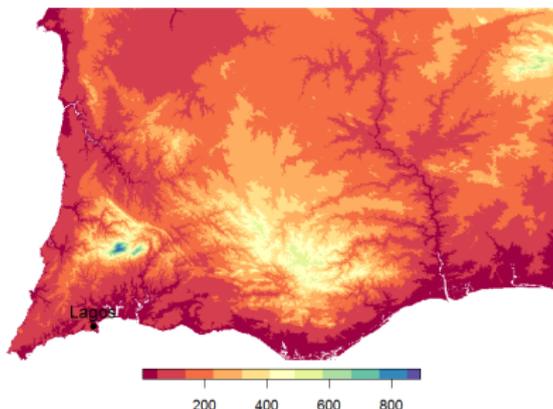
## Análise de dados com package `raster`: Carta hipsométrica, Algarve

Vamos usar a paleta `Spectral` para criar uma carta hipsométrica com 12 classes:

```
1 plot(srtm, xaxt="n", yaxt="n", box=FALSE, axes=FALSE,  
      horizontal=TRUE, col=brewer.pal(n=11, "Spectral"))
```

Adicionar a localização e nome da cidade de Lagos à imagem:

```
1 points(y=37.095753,x=-8.683319,col="white")  
2 text(y=37.095753,x=-8.683319,"Lagos",col="white",pos=3)
```



## Análise de dados com package `raster`. Exercício: construir um mosaico de dados SRTM para Portugal Continental

Complete o código abaixo para criar um modelo digital de elevações SRTM que cubra a maior parte do território continental português.

```
1 HTTP<- "http://dds.cr.usgs.gov/srtm/  
  version2_1/SRTM3/Eurasia/"  
2 for (tile in tiles)  
3 {  
4   nomezip<-paste0(tile, ".hgt.zip")  
5   download.file(url=paste0(HTTP,  
     nomezip), destfile=nomezip, mode=  
     "wb")  
6   unzip(zipfile=nomezip)  
7   srtm<-raster(paste0(tile, ".hgt"))  
8   if (tile==tiles[1]) mde<-srtm else  
     mde<-merge(mde, srtm)  
9 }  
10 plot(mde)
```



## Análise de dados com package `raster`: altimetria

O package `raster` contém a função `terrain` que permite derivar de um modelo digital de elevações um conjunto de variáveis tais como relevo, orientação, direcção de escoamento, índices de rugosidade do terreno, etc.

Também contém a função `hillShade` para derivar a iluminação sobre o terreno a partir de um MDE e da posição da fonte de iluminação.

Por exemplo, a inclinação em radianos pode ser obtida com o seguinte comando (pode ser aplicado a um MDE em coordenadas geográficas desde que a elevação seja em metros):

```
1 inc<-terrain(srtm, opt="slope", units="radians", neighbors=8)
```

Os declives (% , tangente da inclinação) são dados por:

```
1 declives<-100*tan(values(inc)) # em %  
2 range(declives, na.rm=TRUE) # devolve minimo e máximo
```

## Análise de dados com package `raster`: Exercício

O objectivo do exercício é determinar numa imagem a localização do ponto com declive máximo.

Pretende determinar-se na imagem `srtm` usada atrás qual é a localização que tem o maior declive, e em seguida indicar essa localização sobre o mapa.

Depois observa-se a imagem *Google map* para esse local para compreender melhor a razão para esse valor elevado.

Determinar localização do declive máximo usando as coordenadas das células:

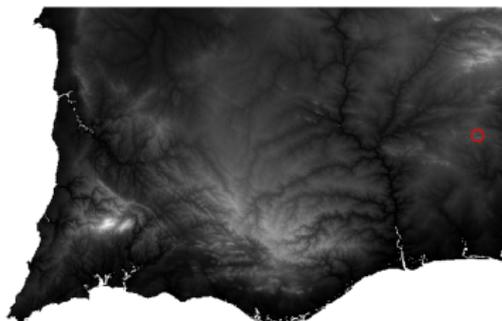
```
1 xyz<-cbind( coordinates( srtm ) , declives )
2 colnames( xyz )<-c( "long" , "lat" , "declive" )
3 # which() é necessário pois declives pode tomar valor NA
4 loc.max<-xyz[ which( declives==max( declives , na.rm=TRUE) ) , ]
```

A matrix `xyz` calculada acima é muito útil para a análise de dados raster, pois combina coordenadas e valores das células.

## Análise de dados com package `raster`: Exercício (cont.)

Construir imagem e localizar ponto:

```
1 plot(srtm, xaxt="n", yaxt="n", box=
  FALSE, axes=FALSE, legend=FALSE,
  col=cores)
2 points(x=loc.max["long"],y=loc.max["
  lat"],col="red",cex=2)
```



Observar o local usando *Google maps*:

```
1 # cria um objecto do tipo extent que é
  um rectangulo em lat/long em torno
  do ponto
2 ext<-extent(c(loc.max[1]-0.003,loc.max
  [1]+0.003,loc.max[2]-0.003,loc.max
  [2]+0.003))
3 # gmap faz download da imagem Google
4 plot(gmap(x=ext,type="hybrid",lonlat=
  TRUE,rgb=FALSE,scale=2) )
```



## Análise de dados com package `raster`: `hillshade`

A função `hillshade` do package `raster` permite calcular a intensidade da luz incidente sobre uma superfície, dispondo de um modelo digital de elevações e sabendo a posição da fonte de iluminação (tipicamente o sol).

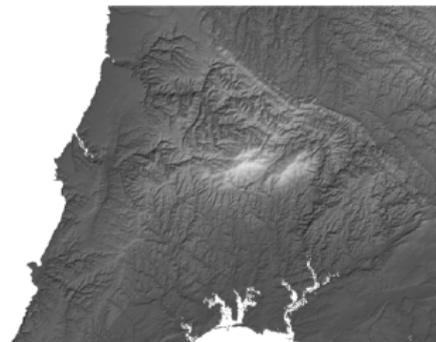
A iluminação depende de `slope` e `aspect`, que são respectivamente as imagens de **inclinação** e de **orientação** de encostas (ambos em radianos).

```
1 inclin<-terrain(srtm,opt="slope",units="radians",neighbors=8)
2 orient<-terrain(srtm,opt="aspect",units="radians",neighbors
  =8)
3 ilumin<-hillShade(slope=inclin,aspect=orient,angle=40,
  direction=150)
```

Os parâmetros `angle` e `direction` determinam a posição do sol. O primeiro é a elevação do sol relativamente ao horizonte (em graus) e o segundo é o azimute (em graus, medido no sentido dos ponteiros do relógio a partir de Norte).

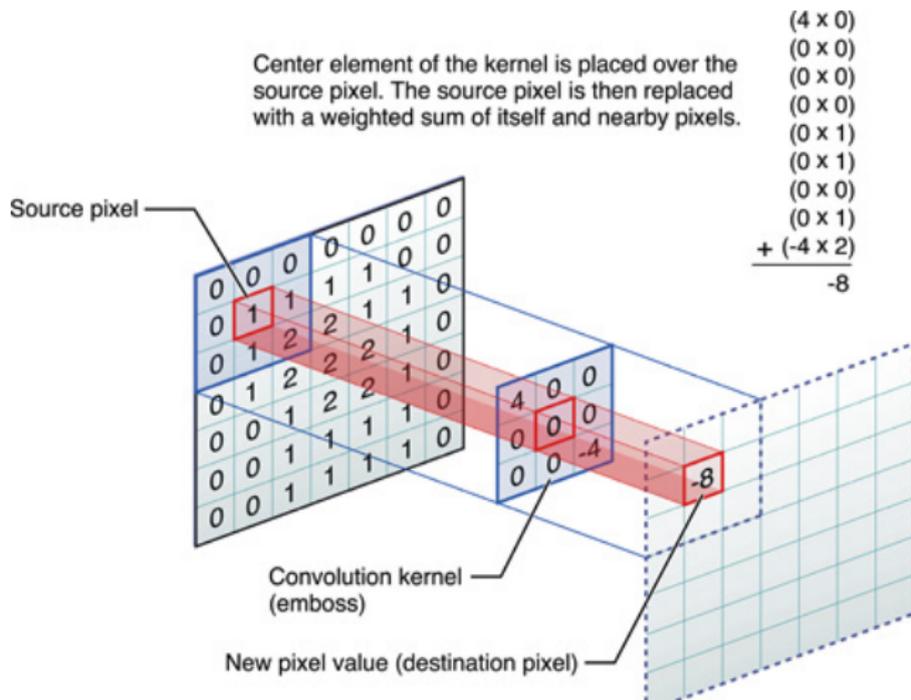
É habitual usar a imagem de iluminação para representar o relevo. Para isso combina-se uma imagem de iluminação com outra de elevações. A imagem de elevações é sobreposta, usando transparência, que é aqui definida pelo parâmetro `alpha` que varia entre 0 (totalmente transparente) e 1 (totalmente opaco).

```
1 extensao<-c(-9, -8.3, 37.1, 37.5)
2 plot(ilumin, ext=extensao, xaxt="n",
3     yaxt="n", box=FALSE, axes=FALSE,
   legend=FALSE, col=cores)
4 plot(srtm, ext=extensao, xaxt="n",
5     yaxt="n", box=FALSE, axes=FALSE,
   legend=FALSE, col=cores, add=TRUE,
6     alpha=.5)
```



## Análise de dados com package `raster`: filtros lineares

Um **filtro linear** é uma técnica de processamento de imagens que consiste em substituir cada célula da imagem por uma combinação linear dos valores das células vizinhas. O **kernel** define a vizinhança e os pesos.



## Análise de dados com package `raster`: filtros lineares

Exemplo: derivar o declive de um MDE usando filtros lineares.

- 1 Em primeiro é necessário re-projectar os dados para as distâncias horizontais serem em metros.
- 2 A diferença de cotas deve ser dividida pela resolução na direcção respectiva para obter a estimativa do declive;
- 3 O declive em cada direcção pode ser definido com os filtros abaixo, em que  $S_x$  é o declive estimado  $S_x$  na direcção W-E, e  $S_y$  é o declive estimado  $S_y$  na direcção S-N.

```
1 srtm.crop<-crop(srtm, extent(c(-8.7, -8.4, 37.1, 37.4))) #  
   Monchique  
2 srtm.etr<- projectRaster(srtm.crop, crs=etr) # para obter  
   coordenadas em metros  
3 Sx <- focal(srtm.etr, w=matrix(c(-1, -2, -1, 0, 0, 0, 1, 2, 1)/(8*res(  
   srtm.etr)[1]), ncol=3))  
4 Sy <- focal(srtm.etr, w=matrix(c(1, 0, -1, 2, 0, -2, 1, 0, -1)/(8*res(  
   srtm.etr)[2]), ncol=3))  
5 declive <- sqrt(Sx^2+Sy^2)
```

## Análise de dados com package `raster`: filtros não lineares

Apesar da função `terrain` ser conveniente, as opções `slope` e `aspect` podem ser definidas à custa de filtros lineares apropriados. Outras opções de `terrain`, tais como `TPI`, `TRI`, `rough`, podem ser obtidos pela aplicação de **filtros não lineares** à imagem:

```
1 f <- matrix(1, nrow=3, ncol=3)
2 # índice de rugosidade do terreno
3 TRI <- focal(srtm, w=f, fun=function(x, ...) sum(abs(x[-5]-x
4     [5]))/8, pad=TRUE, padValue=NA)
5 # índice de posição topográfica
6 TPI <- focal(srtm, w=f, fun=function(x, ...) x[5] - mean(x[-5])
7     , pad=TRUE, padValue=NA)
8 # amplitude das diferenças de elevação
9 rough <- focal(srtm, w=f, fun=function(x, ...) max(x) - min(x),
10     pad=TRUE, padValue=NA, na.rm=TRUE)
```

É possível alterar a dimensão e a expressão do filtro usando os parâmetros `w` e `fun`.

## Análise de dados com package `raster`: filtros não lineares

Exercício: aplicar um filtro de moda ao raster binário

```
ndvi.etr>0.4.
```

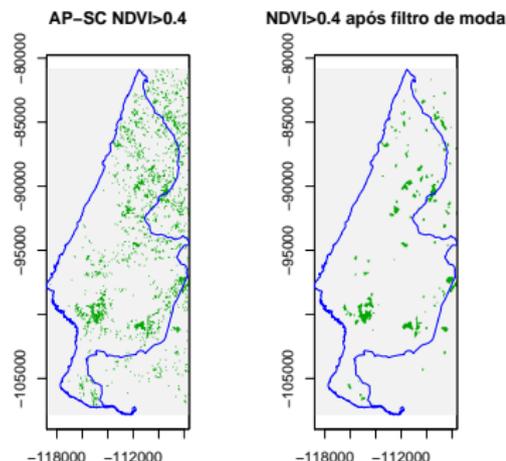
```
1 # filtro não linear a aplicar à imagem:
2 moda<-function(m,na.rm=TRUE)
3 {
4   freqs<-table(m)
5   valores<-as.numeric(names(freqs))
6   return(valores[which(max(freqs)==freqs)])
7 }
8 f<-matrix(1, nrow=3, ncol=3)
9 # focal é lento porque aplica a função moda pixel a pixel:
10 out <- focal(x=ndvi.etr>.4, w=f, fun=moda, pad=TRUE, padValue=
    NA, na.rm=TRUE)
```

Nota: este tipo de filtro é tipicamente usado para remover efeito do tipo *salt-and-pepper* em mapas temáticos, isto é, mapas obtidos por classificação de uma imagem num conjunto de classes (e.g. mapa de ocupação do solo).

## Análise de dados com package `raster`: filtros não lineares

Comparação da imagem antes e depois da aplicação do filtro de moda  $3 \times 3$ .

```
pdf(file="ndvi-maior-do-40-filtro-  
moda.pdf",width=6,height=6)  
par(mfrow=c(1,2));  
plot(ndvi.ets >.4 ,main="AP-SC NDVI  
>0.4",legend=FALSE);  
polygon(limite.ap, border="blue")  
plot(out,main="NDVI>0.4 após filtro  
de moda",legend=FALSE);  
polygon(limite.ap, border="blue")  
graphics.off()
```



No código acima mostra-se como se pode criar uma figura em formato pdf (comando `pdf`) a partir do R. O comando `graphics.off()` fecha a ligação ao ficheiro. Caso se queira criar uma figura com formato png, o comando é `png`.