R e análise espacial para Geologia - Volume 1

André Luiz Lima Costa

“Hubris is the greatest danger that accompanies formal data analysis, including formalized statistical analysis. The feeling of “Give me (or more likely even, give my assistant) the data, and I will tell you what the real answer is!” is one we must all fight against again and again, and yet again.”

John Wilder Tukey

Para Regiane, Gabriela e George. Meus pilares abençoados de minha recuperação pessoal, sou eternamente agradecido a vocês...

Índice

Introdução

Com o avanço tecnológico e a vasta disponibilidade de dados existentes é crucial para o profissional de hoje em dia se familiarizar com métodos e ferramentas novas. O conceito de ‘data science’ é recente mas o cientista de dados já está por ai a bastante tempo. O objetivo desta Apostila é abordar os aspectos envolvidos na manipulação de dados espaciais com R.

Ela cobre na **PARTE 1** aspectos básicos da linguagem R e estatística usando R, como usar e como funciona. Ela dará uma base para seu aprendizado e foi baseada nos sites rspatial.org e r-tutor.com que oferecem de forma concisa uma introdução à linguagem R e estatística.

A **PARTE 2** vai cobrir os tipos de dados espaciais do tipo vector e raster; como são, como ler, e escrever arquivos de dados espaciais, sistemas de coordenadas e como trabalhar com eles num ambiente R.

Na **PARTE 3** veremos os aspectos envolvidos na geoestatística e geoprocessamento utilizando exclusivamente a ferramenta R para o tratamento e análise de dados geológicos. Analisaremos dados espaciais utilizando geoestatística, geoprocessamento e análises de dados geológicos. Usaremos elementos espaciais do tipo pontos, linhas, polígonos e áreas (raster) em duas e três dimensões bem como outros tipos de dados geofísicos.

A melhor forma de armazenar dados é usando um banco de dados integrado a um projeto, onde este seja acessível, atualizável, portável e único e no **VOLUME 2** vamos falar sobre como montar um banco de dados para o armazenamento de informações geoespaciais relacionadas à geologia, mais especificamente ao geoprocessamento e à exploração mineral. Vamos também mostrar como criar rotinas que interagem com o geobanco e outros aplicativos de fonte aberta.

Dados

Os dados usados nesta apostila se encontram em:

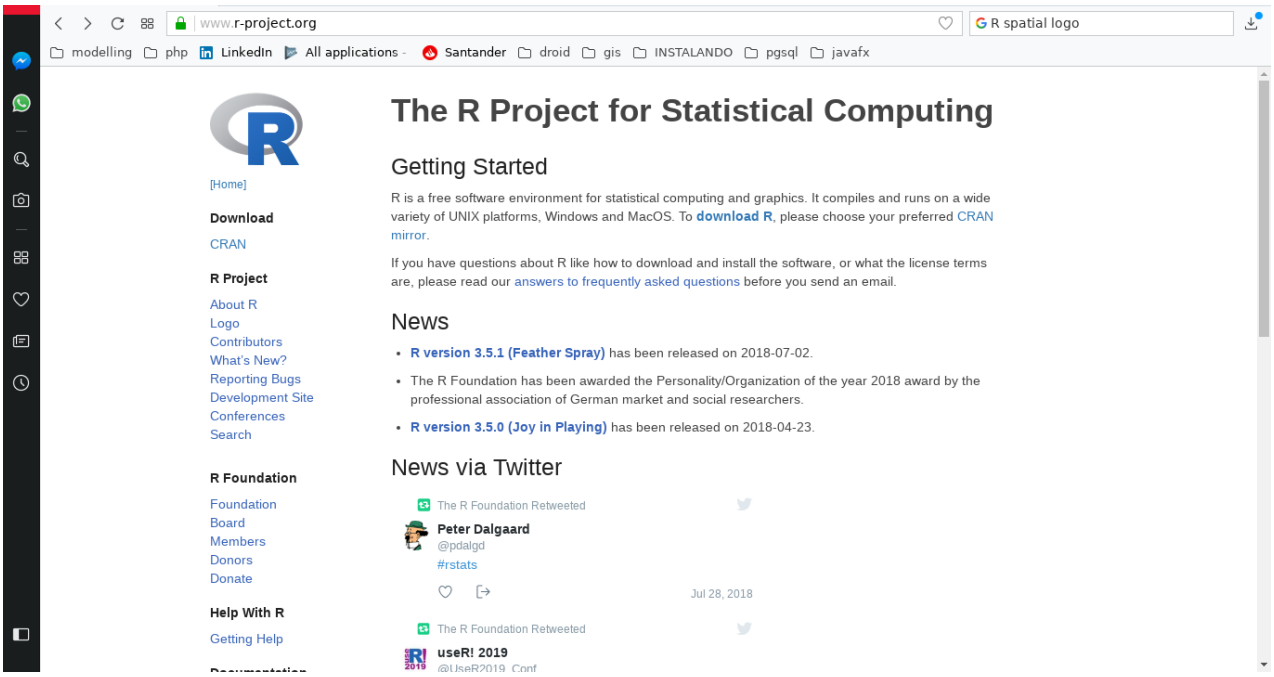
http://amazeone.com.br/barebra/apostilaRgeo/index.php

VISITE:

http://amazeone.com.br

Instalando R

R pode ser instalado facilmente em qualquer sistema operacional através do site da CRAN no www.r-project.org, basta escolher o 'mirror-site' e fazer o download para o seu sistema operacional.



Para iniciar digite R no 'prompt' (aqui representado por: ***[voce@aqui ~]$)*** da sua linha de comando e o seguinte aparecerá. Estamos prontos para começar!

```
[voce@aqui ~]$ R
R version 3.5.0 (2018-04-23) -- "Joy in Playing"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-redhat-linux-gnu (64-bit)
R é um software livre e vem sem GARANTIA ALGUMA.
Você pode redistribuí-lo sob certas circunstâncias.
Digite 'license()' ou 'licence()' para detalhes de distribuição.

R é um projeto colaborativo com muitos contribuidores.
Digite 'contributors()' para obter mais informações e
'citation()' para saber como citar o R ou pacotes do R em publicações.

Digite 'demo()' para demonstrações, 'help()' para o sistema on-line de ajuda,
ou 'help.start()' para abrir o sistema de ajuda em HTML no seu navegador.
Digite 'q()' para sair do R.
```

Prontos para começar!

Fortaleza, 26 de Junho 2020

PARTE 1 – A Linguagem R

1.1 – Tipos básicos

Vamos iniciar aprendendo os mais primitivos tipos de dados e funções auxiliares para cada tipo.

1.1.1 - Tipo numeric

Para criar nossa primeira variável digite:

```
> a <- 1
```

A seta <- foi usada para atribuir o valor 1 à variável 'a', poderíamos usar '=' mas, por convenção, o operador = é usado para assinalar um valor dentro de uma função (veremos mais a frente). O nome 'a' para a variável é arbitrário e poderíamos ter usado b, x, w, raiz, ou qualquer outro nome que nos ajudaria a identificar ele. Existem algumas restrições para o nome de variáveis, elas não podem iniciar com números, não podem ter espaço e nem caracteres especiais tipo '*' mas podem conter" ." .

Para visualizar o valor atribuído a 'a' podemos usar as funções show ou print:

```
>  show(a)
[1] 1
>  print(a)
[1] 1
```

Ou podemos simplesmente digitar o nome da variável:

```
> a
[1] 1
```

IMPORTANTE: Em R todos os tipos básicos são armazenados como um vetor, que é uma array unidimensional com n valores de um certo tipo, mesmo um simples número ou caractere é um vetor.

Podemos usar a função class para checar que tipo é determinado vetor:

```
> class(a)
[1] "numeric"
```

'numeric' significa que 'a' é um número real (decimal). Seu valor é equivalente a 1,000 mas os zeros não são mostrados por padrão. Em alguns casos, criar uma variável do tipo número inteiro (integer) pode ser ser necessário e o fazemos usando uma das duas formas abaixo:

```
> a<- as.integer(9)
> class(a)
[1] "integer"
> a<- 9L
> class(a)
[1] "integer"
```

Para criarmos um vetor com vários valores usamos a função c que combina os valores:

```
> b <- c(3.12, 2.09, 5)
> b
[1] 3.12 2.09 5.00
```

Para criarmos uma sequência regular é mais fácil usar o operador ‘:’ :

```
> s<- 1001:1007
> s
[1] 1001 1002 1003 1004 1005 1006 1007
```

Na ordem reversa:

```
> rs<- 1007:1001
> rs
[1] 1007 1006 1005 1004 1003 1002 1001
```

Podemos criar sequências mais elaboradas usando também a função seq e reverter usando a função rev:

```
> s<- seq(from=6, to=0, by=-0.5)
> s
[1] 6.0 5.5 5.0 4.5 4.0 3.5 3.0 2.5 2.0 1.5 1.0 0.5 0.0
> r<- rev(s)
> r
[1] 0.0 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0
```

Ou repetir o mesmo valor usando rep:

```
> bis<- rep(8,times=4)
> bis
[1] 8 8 8 8
> bis2<- rep(1:2,times=5)
> bis2
[1] 1 2 1 2 1 2 1 2 1 2
> bis3<- rep(1:2, each=5)
> bis3
[1] 1 1 1 1 1 2 2 2 2 2
> bis4<- rep(1:2, each=5,times=3)
> bis4
[1] 1 1 1 1 1 2 2 2 2 2 1 1 1 1 1 2 2 2 2 2 1 1 1 1 1 2 2 2 2 2
```

1.1.2 - Tipo character

O tipo ‘character’ é usado para representar palavras. Também são chamados de ‘string’:

```
> str<-'Altitude Z'
> str
[1] "Altitude Z"
> str<-"Valor 1"
> str
[1] "Valor 1"
```

Usando com função class:

```
> class(str)
[1] "character"
```

Podemos usar **“** ou **‘** mas nunca misturar os dois assim: **str<-“Valor’** .
Criando um vetor de strings:

```
> furos<- c('AT-001','AT-002','AT-003','AT-004')
> furos
[1] "AT-001" "AT-002" "AT-003" "AT-004"
```

A função length informa quantos elementos estão presentes no vetor:

```
> length(furos)
[1] 4
```

A função nchar nos informa o número de caracteres de cada elemento do vetor:

```
> nchar(furos)
[1] 6 6 6 6
```

A função paste une ou concatena strings usando o separador padrão que é o espaço:

```
> lat<- "latitude"
> lon<- "longitude"
> paste(lat,"x",lon)
[1] "latitude x longitude"
```

Podemos alterar o separador usando o argumento sep:

```
> paste(lat,"x",lon,sep='-')
[1] "latitude-x-longitude"
```

Usando o argumento collapse podemos concatenar todos os elementos de um vetor string:

```
> paste(furos,collapse=' / ')
[1] "AT-001 / AT-002 / AT-003 / AT-004"
```

A função substr retorna o intervalo dentro de uma string:

```
> substr("Alvo: Zinco2",1,5)
[1] "Alvo:"
> substr("Alvo: Zinco2",7,12)
[1] "Zinco2"
```

A função sub substitui parte de uma string com outra string:

```
> sub('Zinco','Chumbo',"Alvo: Zinco2")
[1] "Alvo: Chumbo2"
```

A função grep retorna o índice da string num vetor que contém o argumento fornecido:

```
> sondas<- c('LF-230', 'Cs-4002','Cs-4000')
> grep('Cs',sondas)
[1] 2 3
> grep('2',sondas)
[1] 1 2
```

Podemos usar caracteres ‘coringas’ com grep , $ significa termina e ^ significa começa;

```
> grep('2$',sondas)
[1] 2
> grep('^L',sondas)
[1] 1
```

Para ter acesso a cada elemento usamos [n] n representa o número índice e começa com 1:

```
> sondas[1]
[1] "LF-230"
> sondas[1:3]
[1] "LF-230"  "Cs-4002" "Cs-4000"
> sondas[grep('2',sondas)]
[1] "LF-230"  "Cs-4002"
```

1.1.3 - Tipo logical

O tipo lógico representa TRUE ou FALSE (verdadeiro ou falso). São muito utilizados em R e em programação em geral:

```
> verdade <- TRUE
> verdade
[1] TRUE
> class(verdade)
[1] “logical”
```

Valores lógicos são frequentemente o resultado de computação. Por exemplo:

```
> x<- 30
> x<20
[1] FALSE
> x<40
[1] TRUE
```

Valores lógicos podem ser tratados como números TRUE é 1 e FALSE é 0:

```
> v<- 2
> v+TRUE
[1] 3
> v+FALSE
[1] 2
> v*FALSE
[1] 0
> as.logical(1)
[1] TRUE
> as.logical(0)
[1] FALSE
```

1.1.4 - factor

Um fator é uma variável categorizadora com um conjunto de valores chamados levels (níveis). Eles podem ser criados usando a função as.factor. Em R você tipicamente precisa converter uma variável string para um fator para identificar os grupos em modelos e testes estatísticos:

```
> comp<- c('ore','ore','waste','ore','waste','waste','ore','not_defined')
> cat<- as.factor(comp)
> cat
[1] ore         ore         waste       ore         waste       waste
[7] ore         not_defined
Levels: not_defined ore waste
```

Mas números também podem ser usados.

```
> se<- c(1:3,1:5,2:4)
> sec<- as.factor(se)
> sec
 [1] 1 2 3 1 2 3 4 5 2 3 4
Levels: 1 2 3 4 5
```

Vamos agora mover para tipos mais complexos no capítulo seguinte. Pratique e experimente o que vimos neste capítulo.

1.2 – Estrutura de dados básicos

Vimos até agora os tipos básicos de dados em R, números, strings, lógico e fator. Todos estes armazenados em vetor. Agora veremos estruturas adicionais de dados que armazenam dados básicos: Matrix, array, data.frame e list.

1.2.1 - Matrix

O vetor é uma array unidimensional. Uma array bidimensional é representada como uma Matrix. Veja como podemos criar uma matrix com duas linhas e três colunas:

```
> m <- matrix(ncol=3,nrow=2)
> m
     [,1] [,2] [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
> m<- matrix(1:6,ncol=3,nrow=2)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

IMPORTANTE Uma matrix pode ser criada usando as funções cbind (unir colunas) e rbind (unir linhas). Elas são extensivamente usadas em R e peço sua dedicação em entender essas funções.

```
> a<- 1:3
> b<- 4:6
> a
[1] 1 2 3
> b
[1] 4 5 6
```

Usando cbind teremos:

```
> m1<- cbind(a,b)
> m1
     a b
[1,] 1 4
[2,] 2 5
[3,] 3 6
```

Usando rbind teremos:

```
> m2<- rbind(a,b)
> m2
  [,1] [,2] [,3]
a    1    2    3
b    4    5    6
```

Podemos usar cbind e rbind para combinar matrizes, desde que o número de colunas ou linhas dos dois objetos sejam o mesmo:

```
> m3<- cbind(b,b,a)
> m<- cbind(m1,m3)
> m
     a b b b a
[1,] 1 4 4 4 1
[2,] 2 5 5 5 2
[3,] 3 6 6 6 3
```

Podemos acessar informações da estrutura da matrix com as funções nrow, ncol, dim e length:

```
> nrow(m)
[1] 3
> ncol(m)
[1] 5
> dim(m)
[1] 3 5
> length(m)
[1] 15
```

Colunas possuem nomes que podem ser mudados:

```
> colnames(m)
[1] "a" "b" "b" "b" "a"
> colnames(m)<- c('ID','X','Y','Z','V1')
> m
     ID X Y Z V1
[1,]  1 4 4 4  1
[2,]  2 5 5 5  2
[3,]  3 6 6 6  3
```

Da mesma forma as linhas:

```
> rownames(m)<- paste0('linha_',1:nrow(m))
> m
        ID X Y Z V1
linha_1  1 4 4 4  1
linha_2  2 5 5 5  2
linha_3  3 6 6 6  3
```

Uma matrix, assim como um vetor, só pode armazenar um único tipo de dado. Se tentarmos misturar diferentes tipos os mesmos serão convertidos para a categoria que valida todos os dados (muito provavelmente do tipo character):

```
> x<- c(1,2,3)
> x
[1] 1 2 3
> x<- c(x,'a','b')
> x
[1] "1" "2" "3" "a" "b"
> cbind(c('a','b'),1:2)
     [,1] [,2]
[1,] "a"  "1"
[2,] "b"  "2"
```

1.2.2 - Array

Um vetor é uma array unidimensional, uma matriz é uma array bidimensional, acima disso (3 dimensões em diante), denominamos simplesmente array.

Vamos criar agora uma array tridimensional onde colunas são grid na direção x, linhas são grid na direção y e cada array é um nível vertical. Os valores serão digamos teor (como um bloco diagrama 20X20X20, cada elemento seria o valor do teor no bloco):

```
> a1<- c(3,2,0,2,2,1,3,2,1)
> a2<- c(3,3,1,2,2,2,3,3,2)
> a3<- c(4,3,2,2,2,2,4,3,2)
> coluna.nomes<- c(300,340,360)
> linha.nomes<- c(20,40,60)
> matriz.nomes<- c(0,-20,-40)
```

```
> r<- array(c(a1,a2,a3),dim=c(3,3,3),dimnames=list(linha.nomes,coluna.nomes,
matriz.nomes))
> r
, , 0
   300 340 360
20   3   2   3
40   2   2   2
60   0   1   1

, , -20
   300 340 360
20   3   2   3
40   3   2   3
60   1   2   2

, , -40
   300 340 360
20   4   2   4
40   3   2   3
60   2   2   2
```

1.2.3 - list

Uma lista é uma estrutura muito flexível para armazenar dados. Cada elemento de uma lista pode conter qualquer tipo de objeto R (vetor, matrix, array, data.frame, outra lista, etc).

Uma lista simples que mostra que o primeiro elemento da lista [[1]] é o vetor 1 2 3:

```
> list(1:3)
[[1]]
[1] 1 2 3
```

Agora uma lista com dois tipos de dados, string e vetor de números:

```
> l<- list('xyz',c(100,200))
> l
[[1]]
[1] "xyz"

[[2]]
[1] 100 200
```

E uma lista mais complexa contendo uma lista, uma matrix e dois vetores:

```
> m<- matrix(1:6,ncol=3,nrow=2)
> ll<- list(l,m,”abc”,c(0.6,0.5,0.9))
> ll
[[1]]
[[1]][[1]]
[1] “xyz”
[[1]][[2]]
[1] 100 200
[[2]]
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
[[3]]
[1] “abc”
[[4]]
[1] 0.6 0.5 0.9
```

1.2.4 - data.frame

O formato de dado estruturado data.frame é a força da análise estatística de dados em R. Ele é bidimensional mas pode ter colunas de diferente tipos. Um data.frame é o que você obtêm quando importa uma planilha com funções do tipo read.table ou read.csv (que veremos mais adiante).

Podemos também montar um data.frame usando código:

```
> furo<- c('AT-001','AT-002','AT-002','AT-003')
> intervalo<- c(0.1,2.9,3.7,3.6)
> tipo<- c('waste','ore','ore','ore')
> teor<- c(0.04,0.9,1.4,1.4)
> df<- data.frame(furo,intervalo,tipo,teor)
> df
    furo intervalo  tipo teor
1 AT-001       0.1 waste 0.04
2 AT-002       2.9   ore 0.90
3 AT-002       3.7   ore 1.40
4 AT-003       3.6   ore 1.40
```

Funções usadas com data.frame

```
> is.list(df)
[1] TRUE
> names(df)
[1] "furo"      "intervalo" "tipo"      "teor"
> nrow(df)
[1] 4
> ncol(df)
[1] 4
> dim(df)
[1] 4 4
> colnames(df)
[1] "furo"      "intervalo" "tipo"      "teor"
```

Agora que cobrimos os tipos básicos de formatos e estrutura de dados, vamos aprofundar um pouco mais no aprendizado de R. Continue praticando.

1.3 – Índices

Existem várias maneiras de acessar ou substituir valores em vetores ou outras estruturas de dados. A mais comum é usando ‘indexing’ (índice). Também chamado de ‘slicing’ (fatiamento).

1.3.1 - Índice em vetores

Como visto no capítulo 1-1 brevemente, usamos [] para envolver o índice.

Abaixo mostramos alguns exemplos de ‘indexing’

```
> sb<- 1:10
> sb
 [1]  1  2  3  4  5  6  7  8  9 10
> sb[1]
[1] 1
> sb[10]
[1] 10
> sb[4:8]
[1] 4 5 6 7 8
```

Neste outro exemplo excluímos do resultado o elemento de índice 5.

```
> sb[-5]
[1]  1  2  3  4  6  7  8  9 10
```

Podemos também usar índice para modificar valores:

```
> sb[5]<- 64
> sb
 [1]  1  2  3  4 64  6  7  8  9 10
> sb[3:4]<- 999
> sb
 [1]   1   2 999 999  64   6   7   8   9  10
> sb[-1]<- 0
> sb
 [1] 1 0 0 0 0 0 0 0 0 0
```

Uma característica de vetores em R é que vetores menores são reusados quantas vezes forem necessárias para satisfazer o preenchimento do vetor destino:

```
> sb[2:10]<- c(1,2,3)
> sb
 [1] 1 1 2 3 1 2 3 1 2 3
> sb[2:10]<- c(1,2)
Warning message:
In sb[2:10] <-  c(1, 2) :
  número de itens para para substituir não é um múltiplo do comprimento do
substituto
> sb
 [1] 1 1 2 1 2 1 2 1 2 1
```

Note o aviso que o número usado para preencher não é múltiplo dos valores a serem substituídos no segundo caso, mas a substituição é efetuada assim mesmo. Esse comportamento, caso não desejado pode levar a erros na frente.

1.3.2 - Índice em matrix

Como em vetores, elementos de uma matriz podem ser acessados usando índices retornando matrizes ou vetores.

Vamos criar a seguinte matriz 3x3 como exemplo:

```
> m<- matrix(1:9,nrow=3,ncol=3, byrow=TRUE)
> m
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

Existem duas maneiras para acessar um elemento da matriz: usando índice duplo ou simples. O índice duplo nos dá a posição (linha, coluna) e o simples nos dá o enésimo elemento de índice n:

```
> m[2,2]
[1] 5
> m[5]
[1] 5
```

Podemos também obter os seguintes resultados usando índices:

```
> m[1:2,1:2]
     [,1] [,2]
[1,]    1    2
[2,]    4    5
> m[2,]
[1] 4 5 6
> m[,2]
[1] 2 5 8
```

Podemos também acessar colunas pelo seu nome definido conforme abaixo:

```
> colnames(m)<- c('a','b','c')
> m
     a b c
[1,] 1 2 3
[2,] 4 5 6
[3,] 7 8 9
> m[,'b']
[1] 2 5 8
> m[,c('a','c')]
     a c
[1,] 1 3
[2,] 4 6
[3,] 7 9
```

Note que quando acessamos uma coluna ela se transforma automaticamente em um vetor, para manter o formato de coluna podemos usar:

```
> m[,'b',drop=FALSE]
     b
[1,] 2
[2,] 5
[3,] 8
```

Assinalar valores a elementos de uma matrix pode ser feito usando o índice, veja abaixo:

```
> m[1,1]<-  5
> m
     a b c
[1,] 5 2 3
[2,] 4 5 6
[3,] 7 8 9

> m[3,]<-  10
> m
      a  b  c
[1,]  5  2  3
[2,]  4  5  6
[3,] 10 10 10

> m[,2:3] <- 3:1
> m
      a b c
[1,]  5 3 3
[2,]  4 2 2
[3,] 10 1 1
```

O valor da diagonal de uma matrix pode ser assinalado usando a função diag:

```
> diag(m)
[1] 5 2 1
> diag(m)<- 0
> m
      a b c
[1,]  0 3 3
[2,]  4 0 2
[3,] 10 1 0
```

1.3.3 - Índice em list

Índice em lista pode ser um pouco confuso uma vez que você pode se referir a elementos de uma lista ou a elementos de um dado de uma lista (talvez uma matriz). Note a diferença do colchete duplo e do colchete simples.

Ao criarmos a lista abaixo acessamos o terceiro elemento da lista com li[3] mas acessamos a matriz com li[[3]] ou seja, a estrutura da lista é desfeita retornando o elemento na sua forma original:

```
> m<- matrix(1:9,nrow=3,ncol=3,byrow=TRUE)
> colnames(m)<- c('a','b','c')
> li<- list(list(1:3),c('x','y','z','w'),m)
> li[3]
[[1]]
     a b c
[1,] 1 2 3
[2,] 4 5 6
[3,] 7 8 9

> li[[3]]
     a b c
[1,] 1 2 3
[2,] 4 5 6
[3,] 7 8 9
```

Os elementos de uma lista podem ter nomes:

```
> names(li)<- c('lista','vetor','matriz')
> li
$lista
$lista[[1]]
[1] 1 2 3

$vetor
[1] "x" "y" "z" "w"

$matriz
     a b c
[1,] 1 2 3
[2,] 4 5 6
[3,] 7 8 9
```

E seus elementos podem ser extraídos usando esses nomes:

```
> li$vetor
[1] "x" "y" "z" "w"
> li[['matriz']]
     a b c
[1,] 1 2 3
[2,] 4 5 6
[3,] 7 8 9
```

O operador $ também pode ser usado com data.frame (que um tipo especial de list) mas não com matriz.

1.3.4 - Índices em data.frame

Índices em um data.frame podem ser usados como feito em matriz ou lista. Vamos primeiro criar o data.frame a partir da matriz m:

```
> df<- data.frame(m)
> class(df)
[1] "data.frame"
> df
  a b c
1 1 2 3
2 4 5 6
3 7 8 9
```

Note abaixo a diferença quando usamos o índice 1 com colchete duplo ou simples:

```
> df[[1]]
[1] 1 4 7
> df[1]
  a
1 1
2 4
3 7
> df[,1]
[1] 1 4 7
> df[1,]
  a b c
1 1 2 3
```

1.3.5 - Which, %in% e match

Algumas vezes você não sabe o índice dos valores que você precisa e estas funções e operadores o ajudaram a encontrar os índices.

Vejamos como usar which e vetor lógico (TRUE é retornado):

```
> x <- 10:20
> i <- which(x > 15)
> x
 [1] 10 11 12 13 14 15 16 17 18 19 20
> i
[1]  7  8  9 10 11
> x[i]
[1] 16 17 18 19 20

> b <- x>15
> x
 [1] 10 11 12 13 14 15 16 17 18 19 20
> b
 [1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
> x[b]
[1] 16 17 18 19 20
```

Um operador bem útil é o %in% que nos permite perguntar qual valor em um vetor está presente em outro.

Vejamos como:

```
> x<- 10:20
> j<- c(1,3,11,19)
> j %in% x
[1] FALSE FALSE  TRUE  TRUE
> which(j %in% x)
[1] 3 4
```

A função match também nos permite comparar dois vetores e saber os índices dos valores coincidentes.

Note que match(j,x) retorna resultado diferente de match(x,j):

```
> match(j,x)
 [1] NA NA  2 10
> match(x,j)
 [1] NA  3 NA NA NA NA NA NA NA  4 NA
```

Os resultados são:
- o elemento 3 e 4 de j são os mesmos que os elementos 2 e 10 de x.
- o elemento 2 e 10 de x são os mesmos que os elementos 3 e 4 de j.

Cobrimos os principais tipos de dados necessários para esta Apostila. Vamos agora ver como usar eles em R.

1.4 – Álgebra

Vetores e matrizes podem ser usados para computar novos vetores (e matrizes) com expressões algébricas simples e intuitivas.

1.4.1 - Álgebra de vetores

Vamos criar dois vetores:

```
> a <- 1:5
> b <- 6:10
```

Alguns exemplos de como efetuamos operações com esses vetores:

```
> d <- a*b
> a
[1] 1 2 3 4 5
> b
[1]  6  7  8  9 10
> d
[1]  6 14 24 36 50
> a+b
[1]  7  9 11 13 15
> a-b
[1] -5 -5 -5 -5 -5
> b-a
[1] 5 5 5 5 5
> a/b
[1] 0.1666667 0.2857143 0.3750000 0.4444444 0.5000000
> b %% a
[1] 0 1 2 1 0
```

A grande vantagem de R é que não precisamos criar ‘loops’ para efetuar operações em vetores (ou matrizes).

Podemos também efetuar operações de vetores com valores números isolados:

```
> a*3
[1]  3  6  9 12 15
> a/4
[1] 0.25 0.50 0.75 1.00 1.25
> b-5
[1] 1 2 3 4 5
```

1.4.2 - Comparações lógicas

R possui uma série de operadores lógicos:

==	***igual***
!=	***diferente***
>	***maior***
<	***menor***
>=	***maior ou igual***
<=	***menor ou igual***
&	***E (AND) lógico***
|	***OU (OR) lógico***

Vejamos alguns exemplos:

```
> b-5==a
[1] TRUE TRUE TRUE TRUE TRUE
> a!=b
[1] TRUE TRUE TRUE TRUE TRUE
> a > 2
[1] FALSE FALSE  TRUE  TRUE  TRUE
> a < 2
[1]  TRUE FALSE FALSE FALSE FALSE
> b >6 & b < 8
[1] FALSE  TRUE FALSE FALSE FALSE
> b > 9 | a < 2
[1]  TRUE FALSE FALSE FALSE  TRUE
> a >=4
[1] FALSE FALSE FALSE  TRUE  TRUE
```

1.4.3 - Funções matemáticas

R possui uma variedade de funções matemáticas e estatísticas que podem ser usadas para operações com arrays (vetores e matrizes).

Abaixo temos alguns exemplos: sqrt é raiz quadrada, min é o mínimo valor, range é a faixa de valores (min e max), mean é a média, prod é o produto de todos os valores, sd é o desvio padrão e exp é o exponencial:

```
> sqrt(a)
[1] 1.000000 1.414214 1.732051 2.000000 2.236068
> sqrt(b)
[1] 2.449490 2.645751 2.828427 3.000000 3.162278
> min(a)
[1] 1
> min(b)
[1] 6
> range(b)
[1]  6 10
> range(a)
[1] 1 5
> mean(a)
[1] 3
> prod(a)
[1] 120
> sd(a)
[1] 1.581139
> exp(b)
[1]   403.4288  1096.6332  2980.9580  8103.0839 22026.4658
```

1.4.4 - Números aleatórios

É bastante comum a criação de vetores de números aleatórios na análise de dados, e também para criar um modelo ou entender se um procedimento funciona. Para obter 10 números aleatórios amostrados de forma uniforme entre 0 e 1 usamos:

```
> ale<-runif(10)
> ale
 [1] 0.93575073 0.75748417 0.11579646 0.41033991 0.65282156 0.61987880
 [7] 0.73218410 0.03723520 0.78086560 0.06122219
```

Para obtermos números distribuídos de forma Normal (com valores de média e desvio padrão pré-definidos) usamos rnorm:

```
> rno<-rnorm(10,mean=10.4001,sd=0.05005)
> mean(rno)
[1] 10.42109
> sd(rno)
[1] 0.0510869
> rno
 [1] 10.41448 10.38759 10.46205 10.47412 10.38051 10.37194 10.46489 10.47035
 [9] 10.45487 10.33007
```

Se executarmos novamente o comando obteremos diferentes valores, seria o esperado pois são aleatórios! Mas, números gerados em computação não são verdadeiramente aleatórios, mas pseudoaleatórios. Para trabalharmos com análise de dados muitas vezes precisamos de usar os mesmos números aleatórios todas as vezes que repetirmos o experimento ou procedimento. Podemos fazer isso com a função set.seed.

Veja abaixo como funciona:

```
> set.seed(12)
> runif(3)
[1] 0.06936092 0.81777520 0.94262173
> runif(4)
[1] 0.26938188 0.16934812 0.03389562 0.17878500
> runif(5)
[1] 0.641665366 0.022877743 0.008324827 0.392697197 0.813880559
> set.seed(12)
> runif(3)
[1] 0.06936092 0.81777520 0.94262173
> runif(5)
[1] 0.26938188 0.16934812 0.03389562 0.17878500 0.64166537
> set.seed(12)
> runif(3)
[1] 0.06936092 0.81777520 0.94262173
> runif(5)
[1] 0.26938188 0.16934812 0.03389562 0.17878500 0.64166537
```

Note que toda vez que set.seed é chamada a mesma sequência aleatória é executada, conforme queríamos.

O número é arbitrário e uma sequência de números diferente ocorre se mudarmos o número do set.seed:

```
> set.seed(12)
> runif(5)
[1] 0.06936092 0.81777520 0.94262173 0.26938188 0.16934812
> set.seed(120)
> runif(5)
[1] 0.3919077 0.1260168 0.3446131 0.7954961 0.1960155
```

1.4.5 - Álgebra de matrix

A computação de matrizes é também vetorizada, ou seja, dada uma matriz m podemos efetuar m * 5 para multiplicar todos os valores da matriz por 5 ou executar m^2 para elevar a mesma ao quadrado.

Vejamos alguns exemplos:

```
> m<- matrix(1:6, ncol=3,nrow=2,byrow=TRUE)
> m
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> m * 2
     [,1] [,2] [,3]
[1,]    2    4    6
[2,]    8   10   12
> m^2
     [,1] [,2] [,3]
[1,]    1    4    9
[2,]   16   25   36
```

Nós podemos também multiplicar uma matriz por um vetor lembrando que se o número de elementos do vetor for menor o mesmo será reciclado (sentido operação coluna – próxima coluna):

```
> m * c(2,3,4)
     [,1] [,2] [,3]
[1,]    2    8    9
[2,]    12  10   24
```

Podemos multiplicar duas matrizes, mas a multiplicação será célula por célula:

```
> m*m
     [,1] [,2] [,3]
[1,]    1    4    9
[2,]   16   25   36
```

Para usar álgebra linear, ou seja, multiplicação de matrizes usamos %*%.

Nesse caso multiplicando a matriz m pela sua transposta :

```
> m %*% t(m)
     [,1] [,2]
[1,]   14   32
[2,]   32   77
```

Dando prosseguimento, vamos ver como ler e escrever arquivos usando R.

1.5 – Lendo e escrevendo arquivos

Em muitos casos, o primeiro passo na análise de dados é ler arquivos. Isto pode ser bastante complicado devido à grande variação de formatos de arquivos. Aqui mostraremos como ler arquivos do tipo texto no formato planilha que poderão ser colocados numa estrutura do tipo data.frame, que é o procedimento mais comum. Pegamos um arquivo csv ou tab e colocamos na estrutura de dados (data.frame).

Para ler um arquivo, primeiro precisamos saber o seu nome e em qual diretório ele está localizado.

Usamos uma das seguintes formas: a) Colocando essa informação ('path' ou caminho para o arquivo) em uma variável como uma string (não use simples \, use / ou \\ ):

```
> f1 <- "C:/projects/research/data/test.csv"
> f2 <- "C:\\projects\\research\\data\\test.csv"
```

b) Na prática trabalhamos com arquivos localizados no diretório de trabalho corrente, desta forma podemos abrir o arquivo sem o caminho completo, apenas fornecendo o seu nome diretamente.

Obtemos o diretório corrente com a função getwd:

```
> getwd()
[1] "/home/lanjal/workR"
```

Criaremos no diretório de trabalho um arquivo chamado furos.csv com dados sobre furos de sonda. Copie e cole no seu editor de texto e salve como furos.csv:

```
furo,easting,northing,elev,dip,az
AT-001,300300,7989500,400,-90,0
AT-002,300656,7989345,410,-60,123
AT-003,300567,7989789,380,-60,180
AT-004,300123,7989843,350,-45,270
```

Primeiro checamos se o arquivo existe e depois leremos este dado usando:

```
> file.exists('furos.csv')
[1] TRUE
> d <- read.csv('furos.csv', stringsAsFactors=FALSE)
> head(d)
    furo easting northing elev dip  az
1 AT-001  300300  7989500  400 -90   0
2 AT-002  300656  7989345  410 -60 123
3 AT-003  300567  7989789  380 -60 180
4 AT-004  300123  7989843  350 -45 270
> class(d)
[1] "data.frame"
```

Vemos que os dados são carregados automaticamente em um data.frame.

Escrevemos um arquivo a partir de uma data.frame usando:

```
> write.csv(d, 'furos2.csv', row.names=FALSE)
```

Podemos ler um arquivo linha por linha usando:

```
> webpage<- readLines('http://www.r-project.org')
> webpage
 [1] "<!DOCTYPE html>"
 [2] "<html lang=\"en\">"
.
.
.

[123] "  </body>"
[124] "</html>"
```

Trabalhando com arquivos csv é a forma mais eficiente e rápida de importar a informação dentro do ambiente R.

Outros formatos podem ser carregados usando funções específicas para o tal.

No próximo capítulo vamos importar os dados de um arquivo e mostrar funções que nos auxiliam a inspecionar os dados nele contido.

1.6 – Inspecionando os dados

Depois de abrirmos dados de um arquivo, o próximo passo é dar uma inspecionada neles usando algumas funções estatísticas. Primeiro dê uma olhada nos dados em geral para ver se algo estranho ou incoerente aparece. Isso é muito importante pois evitará problemas mais adiante. Vamos ver algumas ferramentas de inspeção.

Criaremos no diretório de trabalho um arquivo chamado amostra.csv com dados sobre análises. Copie e cole no seu editor de texto e salve como amostra.csv:

```
amostra,batch, Au(ppm),Ag(ppm), Cu(ppm)
smp-01,c3,0.03,0.05,1.5
smp-02,c3,0.13,0.15,5.5
smp-02,c3,0.14,0.25,6.5
smp-04,c3,0.10,0.25,9.5
smp-05,c3,0.08,0.24,9.5
smp-06,c3,0.04,0.25,11.5
smp-07,c1,0.03,0.23,8.5
smp-08,c3,0.03,0.22,5.5
smp-08,c3,-999.00,0.05,2.5
smp-10,c3,0.05,0.15,2.5
```

Depois use:

```
> smp <- read.csv('amostra.csv', stringsAsFactors=FALSE)
> smp
   amostra batch Au.ppm. Ag.ppm. Cu.ppm.
1   smp-01    c3    0.03    0.05     1.5
2   smp-02    c3    0.13    0.15     5.5
3   smp-02    c3    0.14    0.25     6.5
4   smp-04    c3    0.10    0.25     9.5
5   smp-05    c3    0.08    0.24     9.5
6   smp-06    c3    0.04    0.25    11.5
7   smp-07    c1    0.03    0.23     8.5
8   smp-08    c3    0.03    0.22     5.5
9   smp-08    c3 -999.00    0.05     2.5
10  smp-10    c3    0.05    0.15     2.5
```

Com a função summary vemos estatística dos dados coluna por coluna (se aplicável):

```
> summary(smp)
   amostra             batch              Au.ppm.           Ag.ppm.
 Length:10          Length:10          Min.   :-999.000   Min.   :0.0500
 Class :character   Class :character   1st Qu.:   0.030   1st Qu.:0.1500
 Mode  :character   Mode  :character   Median :   0.045   Median :0.2250
                                       Mean   : -99.837   Mean   :0.1840
                                       3rd Qu.:   0.095   3rd Qu.:0.2475
                                       Max.   :   0.140   Max.   :0.2500
    Cu.ppm.
 Min.   : 1.50
 1st Qu.: 3.25
 Median : 6.00
 Mean   : 6.30
 3rd Qu.: 9.25
 Max.   :11.50
```

Observamos o seguinte:
- O batch deveria ser c1 para todas as amostras;
- Temos duas amostras 02;
- Temos um valor -999 que deveria ser mudado para NA.

Podemos acessar as colunas do data.frame usando o operador $. Dessa forma podemos mudar os valores conforme necessário.

Mudando o número da amostra:

```
> smp$amostra[3]<- 'smp-03'
> smp$amostra
 [1] "smp-01" "smp-02" "smp-03" "smp-04" "smp-05" "smp-06" "smp-07" "smp-08"
 [9] "smp-08" "smp-10"
```

Mudando o batch:

```
> i<- smp$batch == 'c3' #retorna o(s) índice(s) do elemento do vetor = ‘c3’
> smp$batch[i]<- 'c1'
> smp$batch
 [1] "c1" "c1" "c1" "c1" "c1" "c1" "c1" "c1" "c1" "c1"
```

E mudaremos o valor -999 para NA

```
> smp$Au.ppm.[9]<- NA
> smp$Au.ppm.
 [1] 0.03 0.13 0.14 0.10 0.08 0.04 0.03 0.03   NA 0.05
```

Vendo todos os dados usando novamente summary:

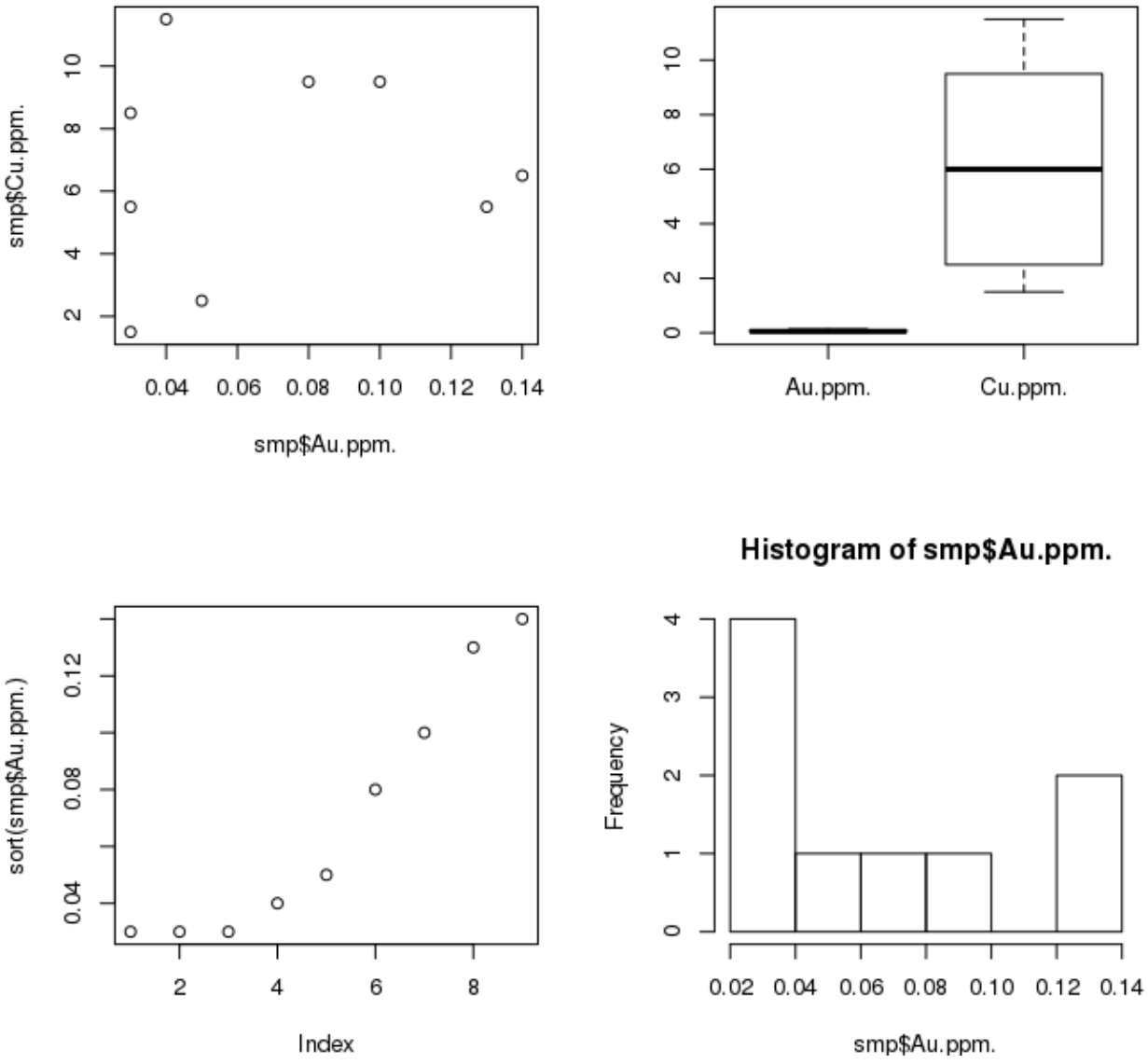
```
> smp
   amostra batch Au.ppm. Ag.ppm. Cu.ppm.
1   smp-01    c1    0.03    0.05     1.5
2   smp-02    c1    0.13    0.15     5.5
3   smp-03    c1    0.14    0.25     6.5
4   smp-04    c1    0.10    0.25     9.5
5   smp-05    c1    0.08    0.24     9.5
6   smp-06    c1    0.04    0.25    11.5
7   smp-07    c1    0.03    0.23     8.5
8   smp-08    c1    0.03    0.22     5.5
9   smp-08    c1      NA    0.05     2.5
10  smp-10    c1    0.05    0.15     2.5
> summary(smp)
   amostra             batch              Au.ppm.        Ag.ppm.
 Length:10          Length:10          Min.   :0.03   Min.   :0.0500
 Class :character   Class :character   1st Qu.:0.03   1st Qu.:0.1500
 Mode  :character   Mode  :character   Median :0.05   Median :0.2250
                                       Mean   :0.07   Mean   :0.1840
                                       3rd Qu.:0.10   3rd Qu.:0.2475
                                       Max.   :0.14   Max.   :0.2500
                                       NA's   :1
    Cu.ppm.
 Min.   : 1.50
 1st Qu.: 3.25
 Median : 6.00
 Mean   : 6.30
 3rd Qu.: 9.25
 Max.   :11.50
```

Agora os dados estão OK.

Vamos ver algumas outras ferramentas gráficas para inspeção visual de dados:

```
> par(mfrow=c(2,2)) #cria área de plotagem de 2 linhas e duas colunas
> plot(smp$Au.ppm.,smp$Cu.ppm.)
> boxplot(smp[,c('Au.ppm.','Cu.ppm.')])
> plot(sort(smp$Au.ppm.))
> hist(smp$Au.ppm.)
```

Gráficos gerados:



Podemos começar a ver o poder de R com relação a análise de dados. Vamos ver agora funções no ambiente R.

1.7 – Funções

Até o momento nós usamos várias funções em R. Por exemplo c, matrix, read.csv e sum. Funções são usadas ou chamadas sempre digitando-se o nome seguido por parêntesis. Muitas vezes, mas nem sempre, fornecemos argumentos dentro dos parêntesis. Se você não digita o parêntesis, as propriedades e definições desta função são descritas.

Vejamos:

```
> nrow
function (x)
dim(x)[1L]
<bytecode: 0x230ca50>
<environment: namespace:base>
```

Nessa descrição vemos que a função nrow usa um único argumento x. Ela chama outra função chamada dim e retorna o seu primeiro elemento no formato inteiro (1L).

Agora vamos ver a função dim que é uma função primitiva de R que usa um argumento:

```
> dim
function (x)  .Primitive("dim")
```

Agora vamos executar a função nrow mas primeiro precisamos de criar o objeto argumento que possui linhas e colunas (um objeto matrix por exemplo).

Criando uma matrix e chamando a função nrow:

```
> m<- matrix(1:6,nrow=2,ncol=3,byrow=TRUE)
> nrow(m)
[1] 2
```

1.7.1 - Escrevendo funções novas

R vem com milhares de funções mas em alguns casos podemos criar funções quando:
- precisamos descrever melhor e de forma mais clara uma tarefa em particular no fluxo de sua análise de dados;
- reutilizar código em vez de repetir sempre os mesmos passos várias vezes;
- criar uma função que é o argumento de uma outra função, é uma prática comum em R.

Escrever as nossas próprias funções não é difícil. Abaixo temos uma bem simples chamada f. Este é um nome arbitrário. Ela não leva nenhum argumento e sempre retorna ‘Use R’.

Criando função f:

```
> f <- function() {
+     return('Use R')
+ }
> f
function() {
    return('Use R')
}
> f()
[1] "Use R"
```

Esta função f é meio sem graça, vamos implementar ela para usar um argumento e retornar esse argumento de alguma forma.

Implementando função f para usar um argumento:

```
> f <-  function(nome) {
+     x<-  paste('Use', nome)
+     return(x)
+ }
> f('R')
[1] "Use R"
> f('GDAL também')
[1] "Use GDAL também"
```

Vamos agora criar uma função que retorna a hipotenusa:

```
> hipotenusa<-function(a,b){
+     h <-  sqrt(a^2+b^2)
+     return(h)
+ }
> hipotenusa(2,5)
[1] 5.385165
> hipotenusa(1,1)
[1] 1.414214
> hipotenusa(3,4)
[1] 5
```

Podemos definir os valores padrões para os argumentos usados:

```
> hipotenusa<-function(a=1,b=1){
+     h <-  sqrt(a^2+b^2)
+     return(h)
+ }
> hipotenusa()
[1] 1.414214
> hipotenusa(8)
[1] 8.062258
> hipotenusa(2,5)
[1] 5.385165
```

A função seguinte é um jogo pedra papel e tesoura em R.

```
> p.p.t<-function(aposta){
+ mine<-floor(runif(n=1,min=1,max=3.9999))
+ print(paste("Sua aposta é",aposta))
+ if(mine==1){
+ print("Minha aposta é pedra")
+ if(aposta=='pedra')print('Empatou')
+ if(aposta=='papel')print('Você ganhou!!!')
+ if(aposta=='tesoura')print('Eu ganhei!!!!!!!!!!')
+ }
+ if(mine==2){
+ print("Minha oposta é tesoura")
+ if(aposta=='pedra')print('Você ganhou!!')
+ if(aposta=='tesoura')print('Empatou')
+ if(aposta=='papel')print('Eu ganhei!!!!!!!!!!')
+ }
+ if(mine==3){
+ print("Minha aposta é papel")
+ if(aposta=='papel')print('Empatou')
+ if(aposta=='tesoura')print('Você ganhou!!')
+ if(aposta=='pedra')print('Eu ganhei!!!!!!!!!!')
```

```
+ }
}
> p.p.t('pedra')
[1] "Sua aposta é pedra"
[1] "Minha aposta é pedra"
[1] "Empatou"
> p.p.t('tesoura')
[1] "Sua aposta é tesoura"
[1] "Minha aposta é pedra"
[1] "Eu Ganhei!!!!!!!!!!!"
> p.p.t('papel')
[1] "Sua aposta é papel"
[1] "Minha aposta é papel"
[1] "Empatou"
```

Um exemplo de uma função que retorna o número de valores únicos de um vetor:

```
> único <-  function(x) {
+     length(unique(x))
+ }
> único(c('a','a','a','b','c','b'))
[1] 3
> único(c(1,1,3,4,5,1))
[1] 4
```

Para obter informações de ajuda sobre uma função existente em R digite ***?*** Antes do nome da função mas sem parêntesis:

```
> ?sum
```

Estes foram apenas alguns exemplos de demonstração de funções em R, você pode criar funções melhores para exercer tarefas mais complexas. Pratique sempre.

1.8 – A Família ‘apply’

Vamos mostrar agora uma família de funções muito importante em R, essa família de funções englobam as funções apply, tapply, aggregate, sapply e lapply.

Estas funções fornecem um meio eficiente e elegante de efetuar operações em porções de uma array ou data.frame, seja em uma coluna ou uma linha em uma matriz e data.frame ou num elemento de uma lista.

Criando uma matrix m:

```
> m <-  matrix(1:15, nrow=5, ncol=3)
> m
     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13
[4,]    4    9   14
[5,]    5   10   15
```

1.8.1 - apply

Cálculo computacional em matrizes é ‘vetorizado’, Por exemplo m * 5 para multiplicar todos os valores da matriz m por 5 ou m ^ 2 para elevar todos os valores ao quadrado. Mas frequentemente precisamos de computar valores somente para partes de uma matriz, uma simples operação para cada linha ou coluna.

A função apply faz isso:

```
> apply(m,1,sum)
[1] 18 21 24 27 30
> apply(m,1,mean)
[1]  6  7  8  9 10
> apply(m,2,sum)
[1] 15 40 65
> apply(m,2,mean)
[1]  3  8 13
```

Observe que apply usa pelo menos 3 argumentos: uma matriz, 1 para linhas ou 2 para colunas, e uma função para calcular o novo valor (ou valores) para cada linha ou coluna conforme especificado em 1 ou 2. Em muitos casos você pode adicionar o parâmetro na.rm=TRUE para remover qualquer NA porque ***em R uma operação que envolve NA sempre vai resultar em NA***.

No exemplo acima usamos as funções sum (soma) e mean (média) mas poderíamos ter usado uma criada por nós mesmo, nesse exemplo nossa função retorna um vetor de valor da soma de cada coluna adicionando o valor um:

```
> maisUm<- function(x){
+ return(sum(x)+1)
+ }
> apply(m,1,maisUm)
[1] 19 22 25 28 31
```

1.8.2 -tapply

Está função pode ser usada para fornecer um sumário estatístico (ex.: média) para grupos de linhas num data.frame. Você precisa de uma coluna que especifique o grupo e assim você pode calcular valores estatísticos para cada grupo.

Criaremos um data.frame onde mostramos os resultados de duas fontes:

```
> colnames(m) <-  c('valor1', 'valor2', 'valor3')
> d <- data.frame(nome=c('Fonte 1', 'Fonte 1', 'Fonte 1', 'Fonte 2', 'Fonte
2'), m, stringsAsFactors=FALSE)
> d$valor2[1]<-NA
> d
     nome valor1 valor2 valor3
1 Fonte 1      1     NA     11
2 Fonte 1      2      7     12
3 Fonte 1      3      8     13
4 Fonte 2      4      9     14
5 Fonte 2      5     10     15
```

Imagine agora que você queira calcular a média para cada valor em cada grupo individual (Fonte):

```
> tapply(d$valor1,d$nome,mean)
Fonte 1 Fonte 2
    2.0     4.5
> tapply(d$valor2,d$nome,mean)
Fonte 1 Fonte 2
     NA     9.5
> tapply(d$valor2,d$nome,mean,na.rm=TRUE)
Fonte 1 Fonte 2
    7.5     9.5
> tapply(d$valor3,d$nome,mean)
Fonte 1 Fonte 2
   12.0    14.5
```

1.8.3 - aggregate

Esta função é similar a tapply mas apresenta o resultado para várias colunas ao mesmo tempo, o segundo argumento deve ser uma lista (não pode ser um vetor)

Veja exemplo com média e desvio padrão:

```
> aggregate(d[, c('valor1', 'valor2', 'valor3')], list(d$nome), mean,
na.rm=TRUE)
  Group.1 valor1 valor2 valor3
1 Fonte 1    2.0    7.5   12.0
2 Fonte 2    4.5    9.5   14.5
> aggregate(d[, c('valor1', 'valor2', 'valor3')], list(d$nome), sd, na.rm=TRUE)
  Group.1    valor1    valor2    valor3
1 Fonte 1 1.0000000 0.7071068 1.0000000
2 Fonte 2 0.7071068 0.7071068 0.7071068
```

1.8.4 - sapply e lapply

Para interagir com uma lista podemos usar lapply ou sapply. A diferença é que lapply retorna uma lista e sapply tenta simplificar o resultado num vetor ou matrix.

Contando o número de caracteres de cada elemento da lista com a função nchar:

```
> minerios <- list('Ouro', 'Prata', 'Urânio', 'Cassiterita', 'Ferro')
> lapply(minerios,nchar)
[[1]]
[1] 4

[[2]]
[1] 5

[[3]]
[1] 6

[[4]]
[1] 11

[[5]]
[1] 5

> sapply(minerios,nchar)
[1]  4  5  6 11  5
```

Pratique bastante essas funções. Vamos a seguir ver controles de fluxo em R.

1.9 – Controles de fluxo

A maioria dos programas possuem dois tipos de controle de fluxo: interação e alternação. Interação é feito via ‘loops’ e alteração via ramificações ‘if-then-else’.

Vamos aqui mostrar rapidamente como usar esses controles de fluxo em R:

1.9.1 - loop-for

Um loop **for** básico é mostrado abaixo. O que o exemplo faz é: para cada valor associado a i entre 1 e 3 executa o que está entre os colchetes {}:

```
> for (i in 1:3) {
+     print('Olá!')
+ }
[1] "Olá!"
[1] "Olá!"
[1] "Olá!"
```

O exemplo abaixo ilustra melhor: soma os valores 1, 2 e 3 e carrega o resultado em j.

```
> j<- 0
> for (i in 1:3){
+ print(i)
+ j<- j+i
+ }
[1] 1
[1] 2
[1] 3
> j
[1] 6
```

Muitas vezes queremos incluir uma condição para parar o loop ou para pular um passo. Isso é feito usando break e next.

Veja o exemplo onde pulamos os valores ímpares e paramos quando passamos 8 no loop que vai até 10:

```
> for (i in 1:10) {
+     if (i %in% c(1,3,5,7)) {
+         next
+     }
+     if (i > 8) {
+         break
+     }
+     print(i)
+ }
[1] 2
[1] 4
[1] 6
[1] 8
```

1.9.2 - loop-while

O loop **while** executa uma tarefa repetidamente até o critério de parada ou break ser encontrado.

Nesse exemplo vemos como funciona o loop: enquanto i < 4 execute:

```
> i <-  0
> while (i < 4) {
+    print(paste(i, 'e contando ...'))
+    i <-  i + 1
+ }
[1] "0 e contando ..."
[1] "1 e contando ..."
[1] "2 e contando ..."
[1] "3 e contando ..."
```

1.9.3 - if-them-else

Ramificações do tipo **if then else** são importantes em qualquer linguagem de programação e não seria diferente em R.

Pegamos duas variáveis x e y e queremos mudar o valor de y dependendo do valor x. Checamos o valor de x se este é igual (==) a cinco, como é verdade (TRUE) o novo valor de y e atribuído:

```
> x<- 5
> y<- 10
> if (x==5){
+ y<- 15
+ }
> y
[1] 15
```

Agora usamos **if-then-else** para testar o valor de x e atribuir o novo valor de y, o último else é executado somente se os parâmetros acima não forem verdadeiros:

```
> if (x > 20) {
+     y <-  y + 2
+ } else if (x > 5 & x < 10) {
+     y <-  y - 1
+ } else {
+     y <-  x
+ }
> y
[1] 5
```

Agora combinando **for** e **if-then-else**:

```
> a <-  1:5
> f <-  vector(length=length(a))
> for (i in 1:length(a)) {
+     if (a[i] > 2) {
+         f[i] <-  a[i] / 2
+     } else {
+         f[i] <-  a[i] * 2
+     }
+ }
> f
[1] 2.0 4.0 1.5 2.0 2.5
```

1.10 – Preparação de dados

Um dos pontos importantes em análise espacial é como organizar os dados existentes, gastamos muito tempo nesta etapa e aqui vamos ver algumas funções em R que nos ajudarão a preparar nossos dados.

Estas funções são reshape e merge.

1.10.1 - reshape

A função reshape nos permite formatar dados usando duas opções: largo para longo e longo para largo. Qual o formato que melhor se adapta à sua análise de dados é escolha sua. Geralmente largo para longo é a melhor opção mas vamos ver as duas abaixo. Copie estes dados abaixo e grave no diretório de trabalho com lab_res.csv.

amostra, batch, lab, au, ag, pt, cu, as, zn, pb
1, 23-A, LHY,0.03,0.09,0.03,0.84,0.30,0.4,0.5
2, 23-A, LHY,0.04,0.07,0.02,0.54,0.35,0.5,0.4
3, 23-A, LHY,0.05,0.04,0.02,0.64,0.34,0.6,0.5
4, 23-A, LHY,0.06,0.01,0.01,0.74,0.27,0.7,0.6
5, 23-A, LHY,0.06,0.02,0.03,0.56,0.14,0.8,0.5
6, 23-A, LHY,0.05,0.03,0.05,0.64,0.21,0.9,0.7
7, 23-A, LHY,0.04,0.05,0.07,0.74,0.34,0.8,0.4
8, 24-A, LHY,0.03,0.09,0.03,0.84,0.56,0.7,0.6
9, 24-A, LHY,0.02,0.07,0.05,0.92,0.35,0.6,0.4
10, 24-A, LHY,0.01,0.06,0.05,0.56,0.36,0.4,0.3
11, 24-A, LHY,0.03,0.05,0.03,0.67,0.32,0.45,0.8
12, 24-A, LHY,0.05,0.06,0.02,0.78,0.54,0.3,0.76
13, 24-A, LHY,0.07,0.07,0.01,0.23,0.34,0.2,0.65
14, 24-A, LHY,0.08,0.08,0.03,0.34,0.24,0.3,0.46
15, 24-A, LHY,0.09,0.09,0.02,0.45,0.34,0.4,0.61
16, 24-A, LHY,0.09,0.10,0.02,0.78,0.23,0.5,0.63

Importe os dados :

```
> lr<- read.csv('lab_res.csv',header=TRUE)
> lr
   amostra batch  lab   au   ag   pt   cu   as   zn   pb
1        1  23-A  LHY 0.03 0.09 0.03 0.84 0.30 0.40 0.50
2        2  23-A  LHY 0.04 0.07 0.02 0.54 0.35 0.50 0.40
3        3  23-A  LHY 0.05 0.04 0.02 0.64 0.34 0.60 0.50
4        4  23-A  LHY 0.06 0.01 0.01 0.74 0.27 0.70 0.60
5        5  23-A  LHY 0.06 0.02 0.03 0.56 0.14 0.80 0.50
6        6  23-A  LHY 0.05 0.03 0.05 0.64 0.21 0.90 0.70
7        7  23-A  LHY 0.04 0.05 0.07 0.74 0.34 0.80 0.40
8        8  24-A  LHY 0.03 0.09 0.03 0.84 0.56 0.70 0.60
9        9  24-A  LHY 0.02 0.07 0.05 0.92 0.35 0.60 0.40
10      10  24-A  LHY 0.01 0.06 0.05 0.56 0.36 0.40 0.30
11      11  24-A  LHY 0.03 0.05 0.03 0.67 0.32 0.45 0.80
12      12  24-A  LHY 0.05 0.06 0.02 0.78 0.54 0.30 0.76
13      13  24-A  LHY 0.07 0.07 0.01 0.23 0.34 0.20 0.65
14      14  24-A  LHY 0.08 0.08 0.03 0.34 0.24 0.30 0.46
15      15  24-A  LHY 0.09 0.09 0.02 0.45 0.34 0.40 0.61
16      16  24-A  LHY 0.09 0.10 0.02 0.78 0.23 0.50 0.63
```

Criaremos um data.frame longo com um elemento e resultado por linha, cada amostra repetirá 7 vezes, uma para cada elemento:

```
> longlr <- reshape(lr,
+   varying = c("au", "ag", "pt", "cu", "as","zn","pb"),
+   v.names = "ppm",
+   timevar= “elem”,
+   times = c("au", "ag", "pt", "cu", "as","zn","pb"),
+   new.row.names = 1:1000,
+   direction = "long")
```

E como resultado temos:

```
> longlr[order(longlr$id),]
    amostra batch  lab elem  ppm id
1         1  23-A  LHY   au 0.03  1
17        1  23-A  LHY   ag 0.09  1
33        1  23-A  LHY   pt 0.03  1
49        1  23-A  LHY   cu 0.84  1
65        1  23-A  LHY   as 0.30  1
81        1  23-A  LHY   zn 0.40  1
97        1  23-A  LHY   pb 0.50  1
2         2  23-A  LHY   au 0.04  2
18        2  23-A  LHY   ag 0.07  2
34        2  23-A  LHY   pt 0.02  2
50        2  23-A  LHY   cu 0.54  2
66        2  23-A  LHY   as 0.35  2
82        2  23-A  LHY   zn 0.50  2
98        2  23-A  LHY   pb 0.40  2
3         3  23-A  LHY   au 0.05  3
19        3  23-A  LHY   ag 0.04  3

.
.
.

16       16  24-A  LHY   au 0.09 16
32       16  24-A  LHY   ag 0.10 16
48       16  24-A  LHY   pt 0.02 16
64       16  24-A  LHY   cu 0.78 16
80       16  24-A  LHY   as 0.23 16
96       16  24-A  LHY   zn 0.50 16
112      16  24-A  LHY   pb 0.63 16
```

Comparando os dois data.frames usando dim:

```
> dim(lr)
[1] 16 10
> dim(longlr)
[1] 112   6
```

Agora vamos transformar este data.frame longo para largo:

```
> larglr <- reshape(longlr,
+   timevar = "elem",
+   idvar = c("id","amostra", "batch", "lab"),
+   direction = "wide")
```

E o resultado é:

```
> larglr
   amostra batch  lab id ppm.au ppm.ag ppm.pt ppm.cu ppm.as ppm.zn ppm.pb
1        1  23-A  LHY  1   0.03   0.09   0.03   0.84   0.30   0.40   0.50
2        2  23-A  LHY  2   0.04   0.07   0.02   0.54   0.35   0.50   0.40
3        3  23-A  LHY  3   0.05   0.04   0.02   0.64   0.34   0.60   0.50
4        4  23-A  LHY  4   0.06   0.01   0.01   0.74   0.27   0.70   0.60
5        5  23-A  LHY  5   0.06   0.02   0.03   0.56   0.14   0.80   0.50
6        6  23-A  LHY  6   0.05   0.03   0.05   0.64   0.21   0.90   0.70
7        7  23-A  LHY  7   0.04   0.05   0.07   0.74   0.34   0.80   0.40
8        8  24-A  LHY  8   0.03   0.09   0.03   0.84   0.56   0.70   0.60
9        9  24-A  LHY  9   0.02   0.07   0.05   0.92   0.35   0.60   0.40
10      10  24-A  LHY 10   0.01   0.06   0.05   0.56   0.36   0.40   0.30
11      11  24-A  LHY 11   0.03   0.05   0.03   0.67   0.32   0.45   0.80
12      12  24-A  LHY 12   0.05   0.06   0.02   0.78   0.54   0.30   0.76
13      13  24-A  LHY 13   0.07   0.07   0.01   0.23   0.34   0.20   0.65
14      14  24-A  LHY 14   0.08   0.08   0.03   0.34   0.24   0.30   0.46
15      15  24-A  LHY 15   0.09   0.09   0.02   0.45   0.34   0.40   0.61
16      16  24-A  LHY 16   0.09   0.10   0.02   0.78   0.23   0.50   0.63
```

Comparando o original com o largo usando dim voltamos a 16 linhas de dados:

```
> dim(larglr)
[1] 16 11
> dim(lr)
[1] 16 10
```

1.10.2 - merge

Podemos unir dois data.frames com mesmo número de linhas com um campo em comum usando merge, primeiro vamos criar os data.frame(s):

```
> furos <-  data.frame(
+     furo = c("At-001", "At-002", "At-003", "At-004", "At-005"),
+     sonda = c("lf", "bs", "lf", "bs", "bs"),
+     completo = c("yes", rep("no", 4)))
> survey <-  data.frame(
+     name = c("At-001", "At-002", "At-003", "At-004", "At-005"),
+     dip = c(-60,-60,-60,-45,-90),
+     az = c(45,270,180,270,0))
```

E agora usamos merge para unir os dois:

```
> mg<- merge(furos, survey, by.x="furo", by.y="name")
> mg
    furo sonda completo dip  az
1 At-001    lf      yes -60  45
2 At-002    bs       no -60 270
3 At-003    lf       no -60 180
4 At-004    bs       no -45 270
5 At-005    bs       no -90   0
```

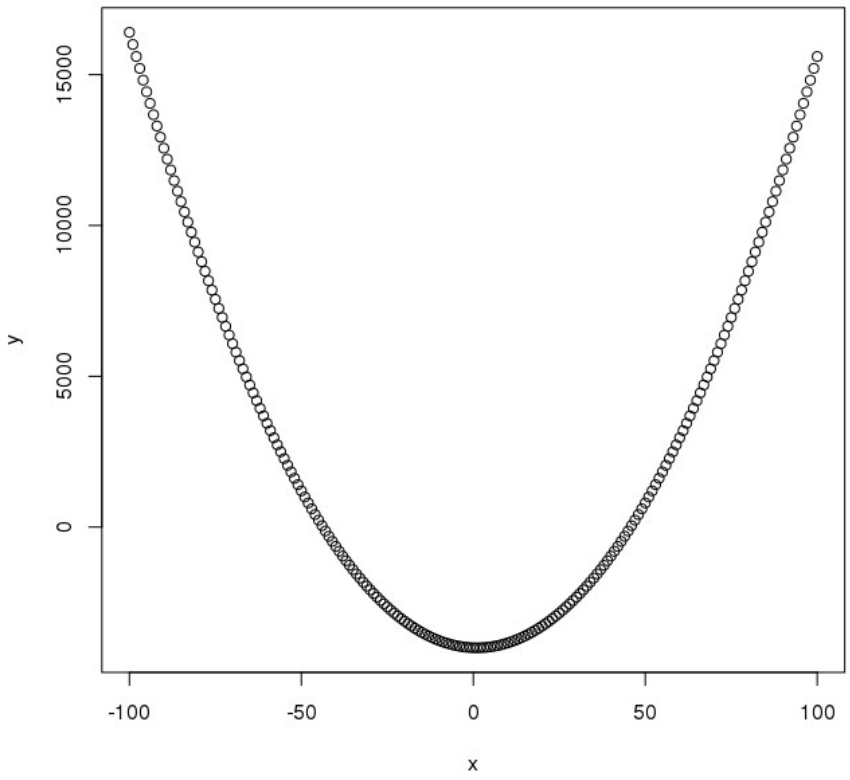
1.11 – Gráficos

Em R podemos gerar diversos tipos de gráficos, vamos aqui mostrar como fazemos isso.

Existem várias maneiras diferentes para fazer gráficos. Vamos aqui lidar com gráficos de pontos usando o pacote “base”.

Vamos utilizar o seguinte dado para criar os primeiros gráficos:

```
> x<-c(-100:100)
> y<-2*x^2-4*x-4000
> plot(x,y)
```



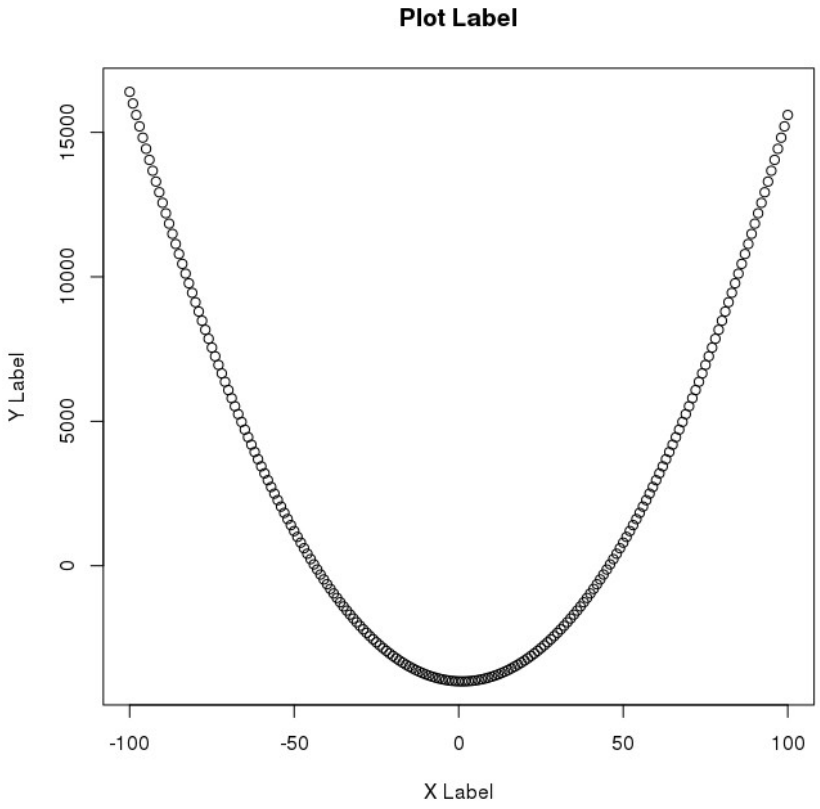
No ambiente R os gráficos são criados em uma nova janela quando chamamos as funções de plotagem.

1.11.1 - Parâmetros da função plot

A função plot tem vários parâmetros. Vamos ver alguns deles.

Parâmetros de rótulos. Use main para rotular o gráfico. Para rotular o eixo X use xlab e use ylab para rotular o eixo Y.

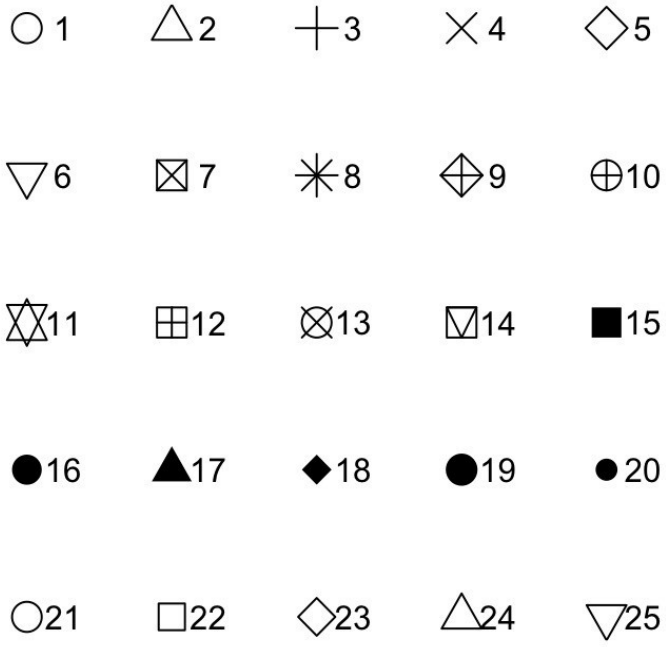
```
> plot(x,y,xlab='X Label',ylab='Y Label',main='Plot Label')
```



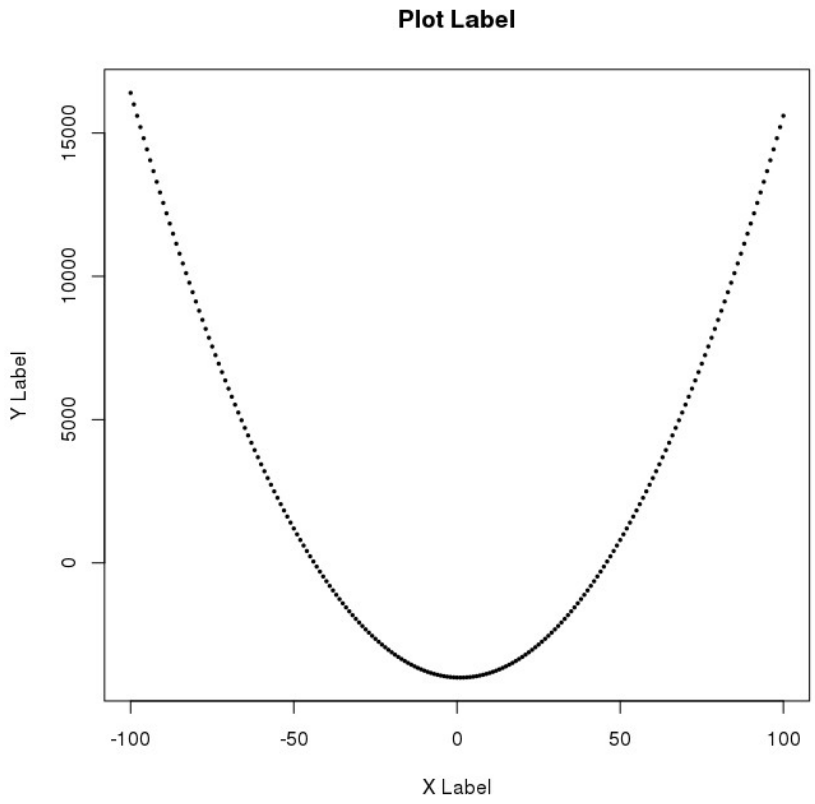
Para fechar a janela podemos usar a função dev.off:

```
> dev.off()
null device
          1
```

Use os parâmetros cex para definir o tamanho do símbolo a ser usado e pch para mudar o formato do símbolo. Veja abaixo uma tabela com os pch dos símbolos.

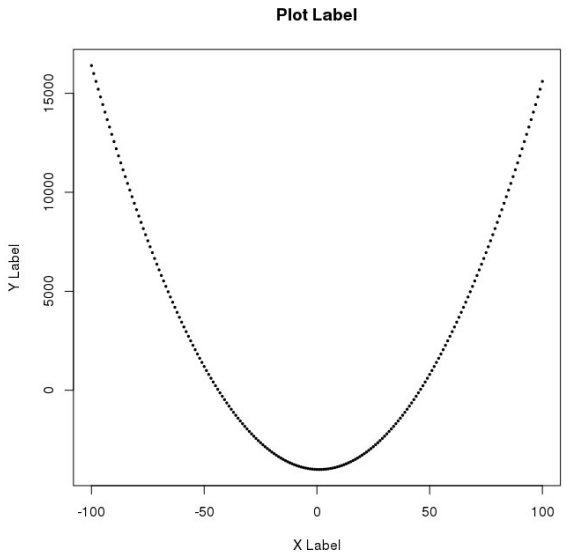


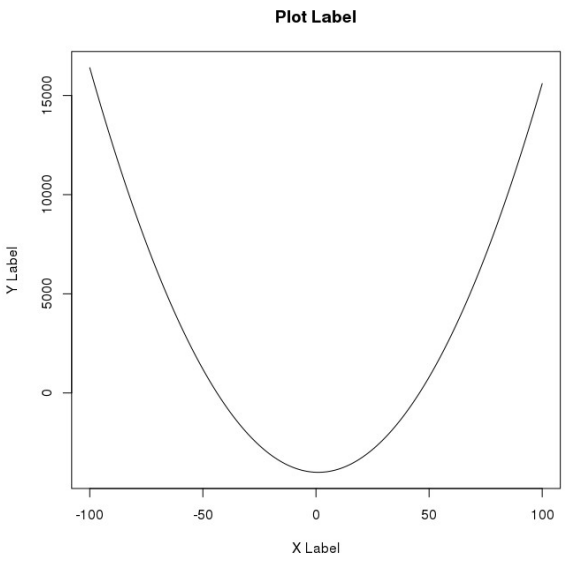
```
> plot(x,y,xlab='X Label',ylab='Y Label',main='Plot Label',pch=20,cex=0.5)
```

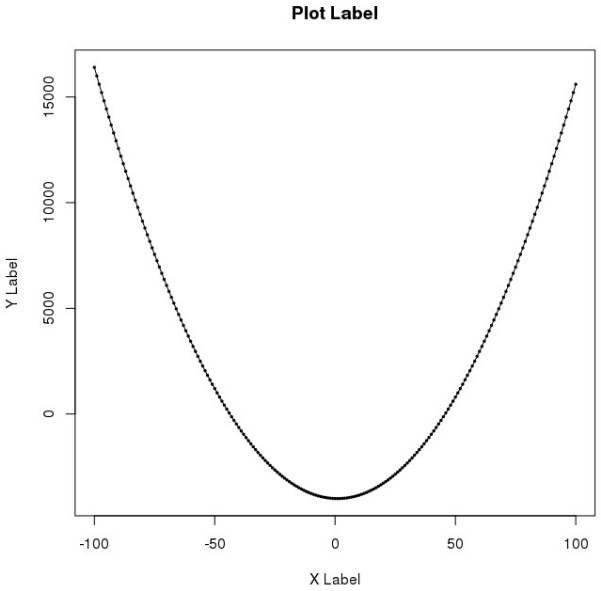


Use o parâmetro type para selecionar entre símbolo, linha ou os dois.

```
> plot(x,y,xlab='X Label',ylab='Y Label',main='Plot ',pch=20,cex=0.5,type= 'p')
> plot(x,y,xlab='X Label',ylab='Y Label',main='Plot ',pch=20,cex=0.5,type= 'l')
> plot(x,y,xlab='X Label',ylab='Y Label',main='Plot ',pch=20,cex=0.5,type= 'o')
```

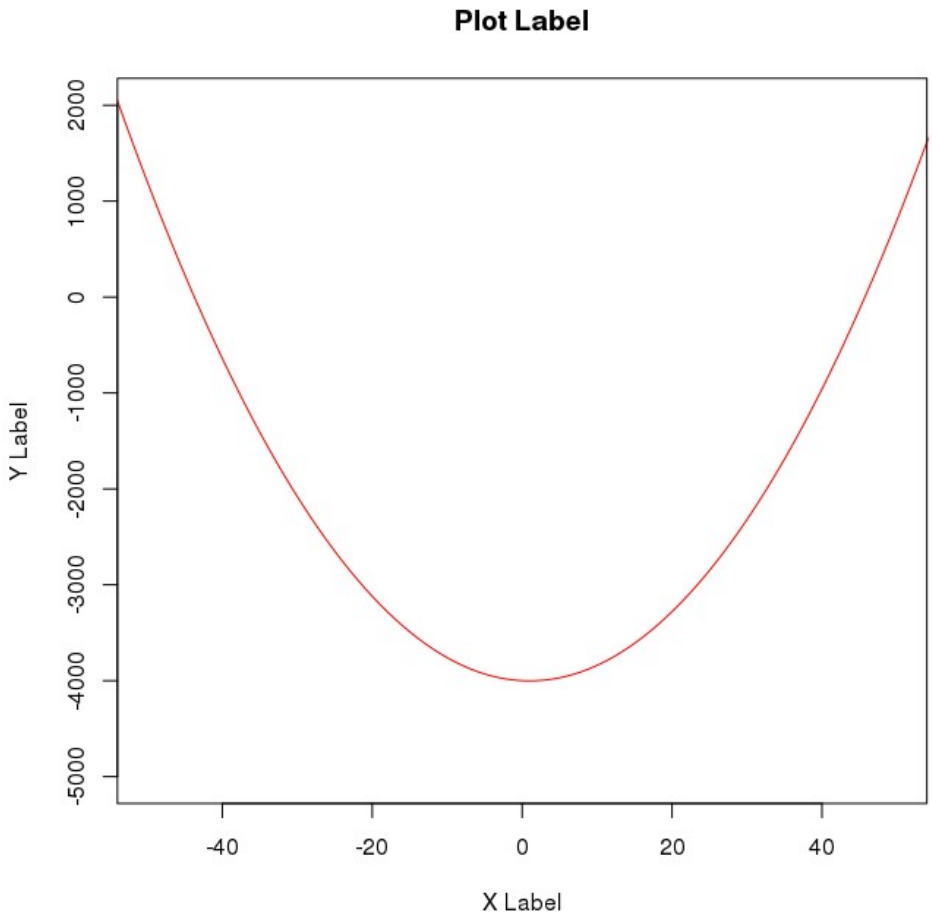






Use col para definir uma cor e xlim e ylim para definir a faixa de X e Y a ser plotada.

```
> plot(x,y,xlab='X Label',ylab='Y Label',main='Plot Label',pch=20, cex=0.5,
type='l',col='red',xlim=c(-50,50),ylim=c(-5000,2000))
```

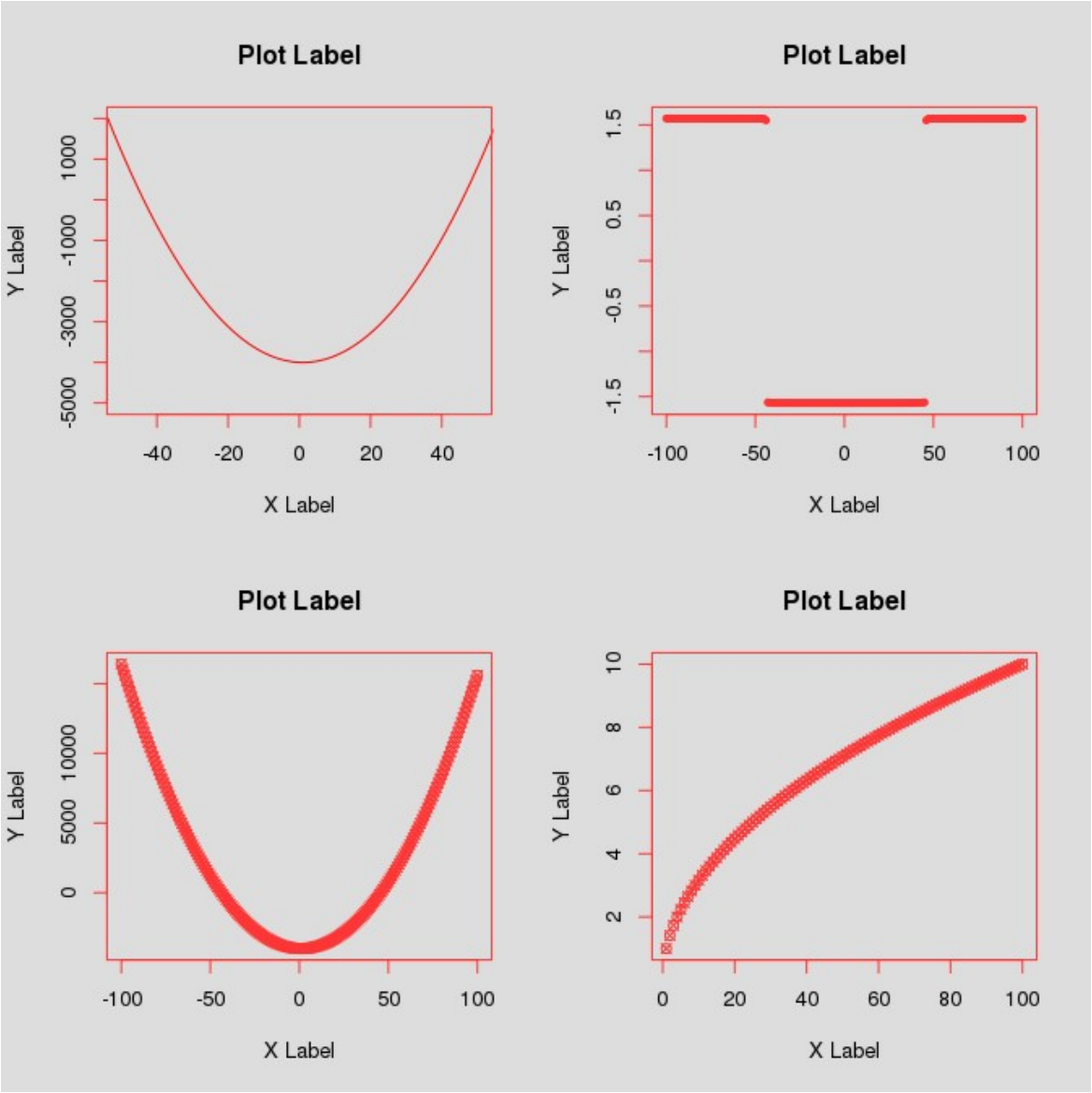


1.11.2 - Parâmetros da função par

A função par prepara a janela de plotagem. Existem vários ajustes nela que podem ser manipulados. Vamos ver alguns deles:

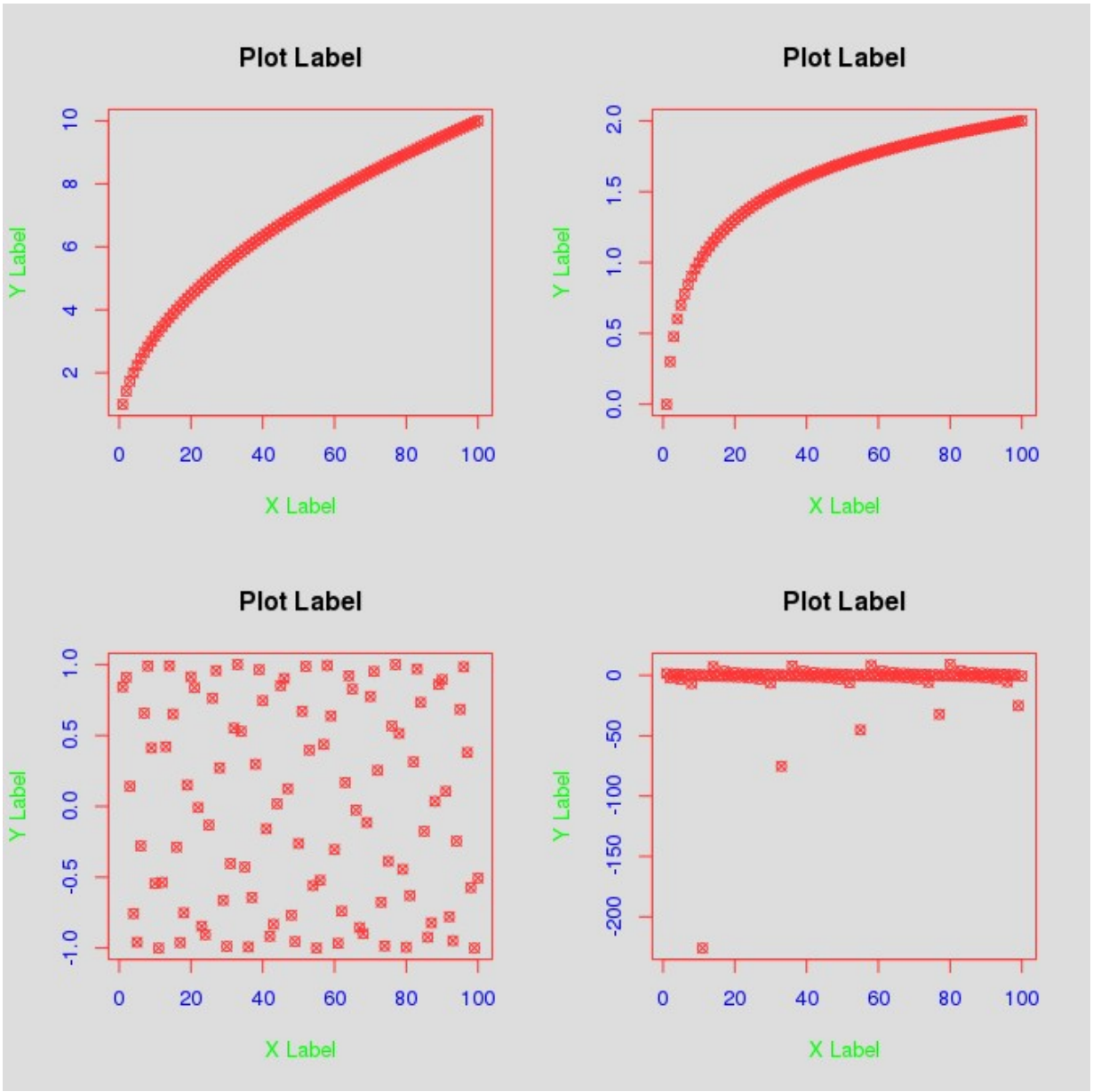
O parâmetro mfrow define o número de subplots organizados em linhas e colunas e o parâmetro bg e fg definem as cores de fundo e frente. Para fechar a janela de plotagem ao seu original use dev.off()

```
> par(mfrow=c(2,2),bg='#dddddd',fg='#ff3434')
> plot(x,y,xlab='X Label',ylab='Y Label',main='Plot Label',pch=20, cex=0.5,
type='l',
col='red',xlim=c> plot(x,atan(y),xlab='X Label',ylab='Y Label',main='Plot
Label',pch=20)
> plot(x,y,xlab='X Label',ylab='Y Label',main='Plot Label',pch=13)
> plot(c(1:100),sqrt(c(1:100)),xlab='X Label',ylab='Y Label',main='Plot Label',
pch=13)
```



Mudando as cores de rótulos e eixos usando par

```
> par(mfrow=c(2,2),bg='#dddddd',fg='#ff3434',col.axis='blue',col.lab='green')
> plot(c(1:100),sqrt(c(1:100)),xlab='X Label',ylab='Y Label',main='Plot Label',
pch=13)
> plot(c(1:100),log10(c(1:100)),xlab='X Label',ylab='Y Label',main='Plot
Label', pch=13)
> plot(c(1:100),sin(c(1:100)),xlab='X Label',ylab='Y Label',main='Plot Label',
pch=13)
> plot(c(1:100),tan(c(1:100)),xlab='X Label',ylab='Y Label',main='Plot Label',
pch=13)
```



1.12 – Modelos estatísticos

Neste capítulo vamos fazer um ‘curso relâmpago’ de estatística elementar. Vamos cobrir bem superficialmente todos os pontos da estatística básica usando R (fonte: www.r-tutor.com/elementary-statistics, modificado).

Primeiramente vamos importar a biblioteca (library) MASS que contém os data.frames necessários:

```
> library(MASS)
```

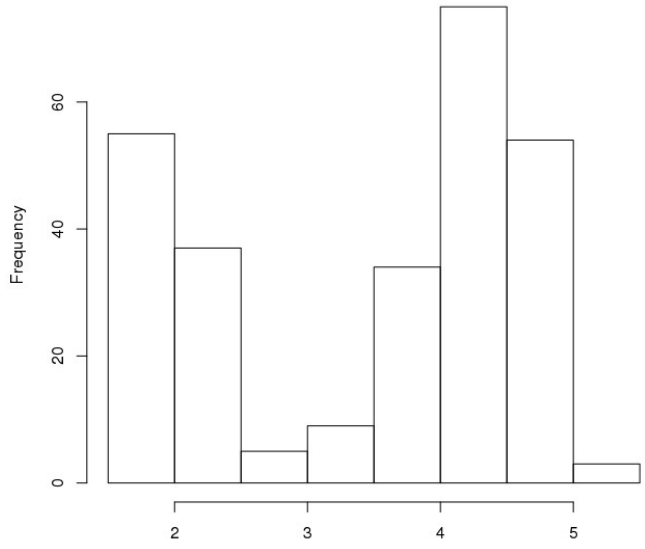
1.12.1 - Dados quantitativos

A frequência de distribuição de um dado é um resumo da ocorrência destes em categorias não sobrepostas. Ou seja, pegamos a distribuição total destes dados e distribuímos em faixas.

```
> duracao<- faithful$eruptions #carrega a duração das erupções
> range(duracao) # exibe a faixa de duração das erupções
[1] 1.6 5.1
> quebar = seq(1.5,5.5,by=0.5) #cria a quebra em intervalos de 0.5
> duracao.dividido<-cut(duracao,quebar,right=FALSE) #classifica as erupções nas
quebras
> duracao.freq<- table(duracao.dividido) # cria as faixas de frequência dos
dados
> cbind(duracao.freq) #mostra na forma de coluna
        duracao.freq
[1.5,2)           51
[2,2.5)           41
[2.5,3)            5
[3,3.5)            7
[3.5,4)           30
[4,4.5)           73
[4.5,5)           61
[5,5.5)            4
```

Ou podemos criar diretamente o histograma com:

```
> hist(faithful$eruptions)
```



A frequência relativa é a frequência dividida pelo tamanho da amostragem.

Assim a frequência relativa seria:

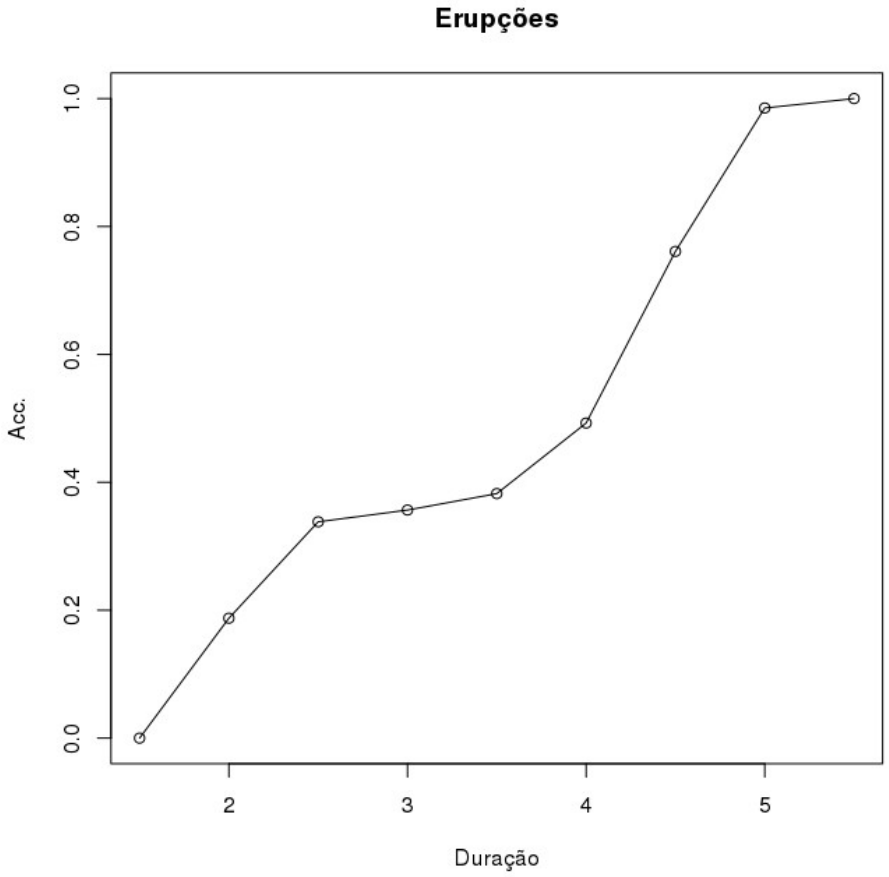
```
> duracao.freqrel<- duracao.freq/nrow(faithful)
> old<- options(digits=1)
> cbind(duracao.freq,duracao.freqrel)
        duracao.freq duracao.freqrel
[1.5,2)           51            0.19
[2,2.5)           41            0.15
[2.5,3)            5            0.02
[3,3.5)            7            0.03
[3.5,4)           30            0.11
[4,4.5)           73            0.27
[4.5,5)           61            0.22
[5,5.5)            4            0.01
```

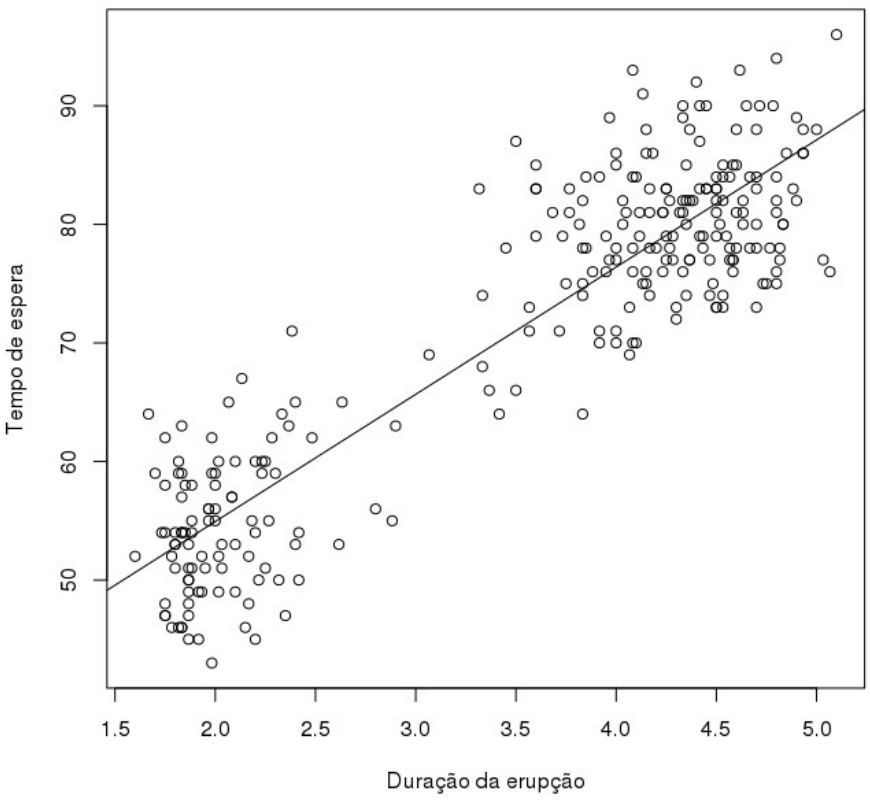
Agora veremos estas mesmas frequências distribuídas na forma acumulada.

```
> duracao.freqcum<- cumsum(duracao.freq)
> cbind(duracao.freqcum)
        duracao.freqcum
[1.5,2)              51
[2,2.5)              92
[2.5,3)              97
[3,3.5)             104
[3.5,4)             134
[4,4.5)             207
[4.5,5)             268
[5,5.5)             272

> duracao.freqrelcum<- duracao.freqcum/nrow(faithful)
> old<- options(digits=2)
> cbind(duracao.freqcum,duracao.freqrelcum)
        duracao.freqcum duracao.freqrelcum
[1.5,2)              51               0.19
[2,2.5)              92               0.34
[2.5,3)              97               0.36
[3,3.5)             104               0.38
[3.5,4)             134               0.49
[4,4.5)             207               0.76
[4.5,5)             268               0.99
[5,5.5)             272               1.00
```

Criando o gráfico das frequências relativas acumuladas e gráfico de erupções x espera com ‘trend’ de regressão linear:

```
> cumrelfreq<- c(0,duracao.freqrelcum)
> plot(quebra,cumrelfreq,main="Erupções",xlab="Duração",ylab="Acc.")
> lines(quebra,cumrelfreq)
> plot(faithful[1:2],xlab="Duração da erupção",ylab="Tempo de espera")
> abline(lm(faithful$waiting~faithful$eruptions))
```


Erupções
Acc.
0.0
0.2
0.4
0.6
0.8
1.0
2
3
4
5
Duração


Tempo de espera
50
60
70
80
90
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Duração da erupção

1.12.2 - Medidas numéricas

Média é a soma de todos os valores dividido pelo número de amostragem:

```
> mean(duracao)
[1] 3.487783
```

Mediana é o valor central dos dados organizados de forma ascendente:

```
> median(duracao)
[1] 4
```

Quartis são os valores que cortam os 25%, 50% (mediana) , 75% e 100% dos dados.

```
> quantile(duracao)
     0%     25%     50%     75%    100%
1.60000 2.16275 4.00000 4.45425 5.10000
```

Percentil: valor que corta n% dos dados.

```
> quantile(duracao,c(.05,.95))
   5%   95%
1.800 4.817
```

Extensão é o valor da diferença entre o maior e o menor valor

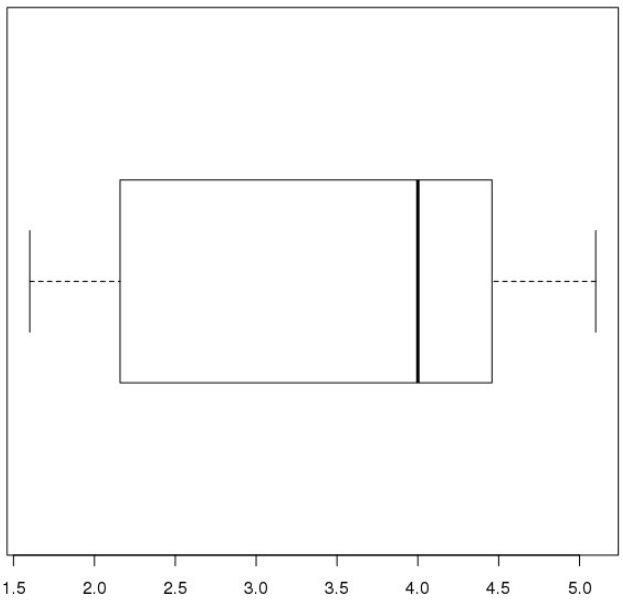
```
> max(duracao)-min(duracao)
[1] 3.5
```

Extensão interquartil é a diferença entre o valor do quartil 75% menos o valor do quartil 25%:

```
> IQR(duracao)
[1] 2.2915
```

Boxplot é a representação dos quartis e da extensão dos dados

```
> boxplot(duracao, horizontal=TRUE)
```



Variância é a medida de como os valores estão dispersos em volta da média;

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

```
> var(duracao)
[1] 1.302728
```

Desvio padrão é a raiz quadrada da variância:

```
> sd(duracao)
[1] 1.141371
```

A covariância de duas variáveis num conjunto de dados mede como estes estão linearmente relacionados.

$$s_{xy} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

Uma covariância positiva indica uma correlação positiva entre as duas variáveis:

```
> cov(duracao,faithful$waiting)
[1] 13.97781
```

O coeficiente de correlação entre duas variáveis em um conjunto de dados é igual a covariância dividida por seus desvios padrão. É uma medida normalizada de como as duas variáveis são linearmente relacionadas.

$$r_{xy} = \frac{s_{xy}}{s_x s_y}$$

Um coeficiente próximo a 1 indica que as duas variáveis são positivamente relacionadas.

```
> cor(duracao,faithful$waiting)
[1] 0.9008112
```

O momento central pode ser calculado usando a seguinte fórmula:

$$m_k = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^k$$

Em particular, o segundo momento central de uma amostragem é praticamente a sua variância, abaixo calculamos o momento central de terceira ordem:

```
> library(e1071)
> moment(duracao, order=3, center=TRUE)
[1] -0.6149059
```

Skewness (assimetria ou obliquidade) é uma medida de simetria dos dados. Como regra, valores negativos indicam média menor que a mediana (mais para esquerda) e valores positivos indicam média maior que a mediana (mais para direita).

$$\gamma_1 = \mu_3 / \mu_2^{3/2}$$

Encontrando o skewness:

```
> library(e1071)
> skewness(duracao)
[1] -0.4135498
```

Curtose é uma medida de forma que caracteriza o achatamento da curva da função de distribuição de probabilidade, se negativo é chamada platicúrtica (achatada), se 0 é mesocúrtica (normal) e se positiva é chamada leptocúrtica (comprida):

$$\gamma_2 = \mu_4 / \mu_2^2 - 3$$

Achando a curtose:

```
> library(e1071)
> kurtosis(duracao)
[1] -1.511605
```

1.12.3 - Distribuição probabilística

A **distribuição binomial** é uma distribuição de probabilidade discreta. Ela descreve o resultado de tentativas em um experimento e cada tentativa tem dois resultados possíveis (binomial). Podemos expressar pela função abaixo:

$$f(x) = \binom{n}{x} p^x (1-p)^{(n-x)} \quad where\ x = 0, 1, 2, ..., n$$

Qual seria a probabilidade de termos 4 respostas certas em 12 tentativas onde cada tentativa temos 20% de acerto (5 opções)? Em R calculamos assim e temos 92.7% de chances de acertarmos 4 vezes:

```
> pbinom(4,size=12,prob=0.2)
[1] 0.9274445
```

A **distribuição de Poisson** é a distribuição de probabilidade da ocorrência de eventos independentes em um intervalo. A função abaixo mostra essa distribuição onde λ é a média:

$$f(x) = \frac{\lambda^x e^{-\lambda}}{x!} \quad where\ x = 0, 1, 2, 3, ...$$

Temos 12 carros cruzando uma ponte em média, qual a probabilidade de termos 17 carros cruzando a ponte em determinado minuto? Em R calculamos assim e temos como resposta 10,13% de chances disso acontecer:

```
> ppois(16, lambda=12, lower=FALSE)
[1] 0.101291
```

A **distribuição contínua uniforme** é a distribuição de probabilidade de números aleatórios selecionados de um intervalo contínuo entre a e b. Sua função é definida por:

$$f(x) = \begin{cases} \frac{1}{b-a} & when\ a \leq x \leq b \\ 0 & when\ x < a\ or\ x > b \end{cases}$$

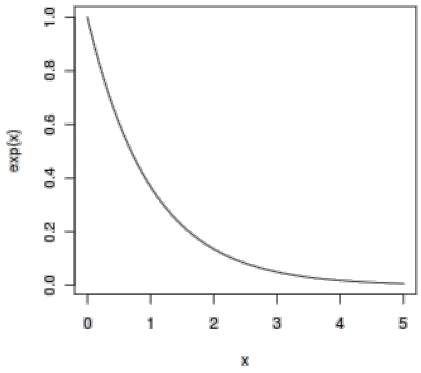
Em R geramos 8 números aleatórios num intervalo (1 e 3) usando:

```
> runif(8, min=1, max=3)
[1] 1.578376 1.893886 2.518432 2.450899 1.095584 1.096872 1.046974 2.052164
```

A **distribuição exponencial** descreve o tempo de chegada de uma sequência de eventos independentes aleatoriamente recorrentes. Se μ é a media do tempo de espera para o próximo evento recorrer, sua função probabilística será:

$$f(x) = \begin{cases} \frac{1}{\mu} e^{-x/\mu} & when\ x \geq 0 \\ 0 & when\ x < 0 \end{cases}$$

Abaixo está um gráfico de distribuição exponencial com μ=1:



Suponhamos que o tempo que um caixa leva para cobrar uma compra seja 3 minutos, qual seria a probabilidade do caixa cobrar em menos de 2 minutos? Em R calculamos assim (1/3 de compras por minuto), a resposta seria 48.7% de chances de concluir a compra em menos de 2 minutos:

```
> pexp(2,rate=1/3)
[1] 0.4865829
```

A **distribuição normal** é definida pela seguinte função, onde μ é média da população e σ^2 é a variância:

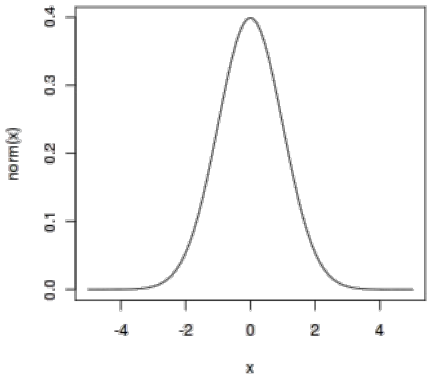
$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2}$$

Se uma variável aleatória X segue uma distribuição normal, escrevemos:

$$X \sim N(\mu, \sigma^2)$$

A distribuição normal é importante devido ao **Teorema do Limite Central**, que diz que a população de todas as amostras possíveis de tamanho n de uma população de média μ e variância σ^2 se aproxima da distribuição normal com média μ e σ^2/n quando n se aproxima do infinito.

Abaixo está um gráfico de distribuição normal com μ=0 e σ=1:



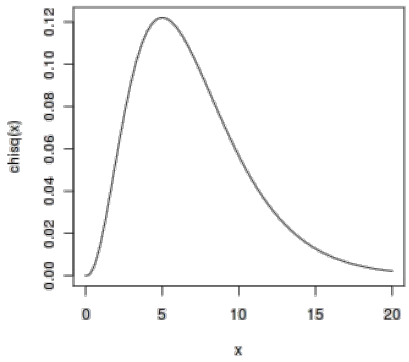
Vamos assumir que as notas de entrada num curso obedeça a distribuição normal e a média de entrada seja 72 e o desvio padrão 15.2. Qual seria a porcentagem de estudantes que teriam nota acima de 84? Em R resolvemos assim e vemos que 21,5% teriam nota acima de 84:

```
> pnorm(84, mean=72, sd=15.2, lower.tail=FALSE)
[1] 0.2149176
```

Se $X_1, X_2, \ldots, X_m$ são m variáveis aleatórias independentes em uma distribuição normal padrão então a quantidade seguinte segue uma **distribuição Chi quadrática** com m graus de liberdade. Sua média será m e sua variância 2m.

$$V = X_1^2 + X_2^2 + \cdots + X_m^2 \sim \chi^2_{(m)}$$

Abaixo temos um gráfico de uma distribuição chi quadrática com 7 graus de liberdade:



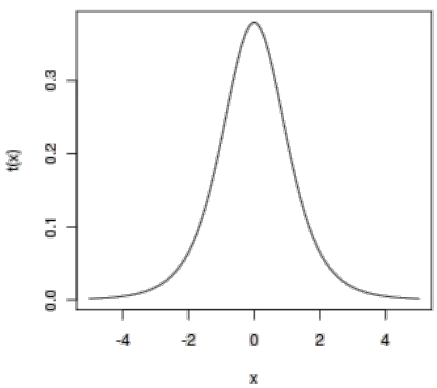
Encontre o percentil 95 de uma distribuição chi quadrática com 7 graus de liberdade. Em R acharíamos assim:

```
> qchisq(.95, df=7)
[1] 14.06714
```

Assumindo que uma variável aleatória Z tem distribuição normal padrão e outra variável V tem distribuição chi quadrática com m graus de liberdade e assumindo que as duas são independentes, então a quantidade seguinte segue uma **distribuição T student** com m graus de liberdade.

$$t = \frac{Z}{\sqrt{V/m}} \sim t_{(m)}$$

Abaixo temos um gráfico de uma distribuição T student com 5 graus de liberdade:



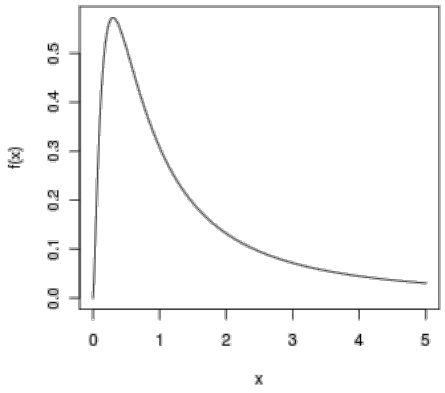
Encontre o percentil 2.5 e o 97.5 de uma distribuição student com 5 graus de liberdade. Em R achamos assim:

```
> qt(c(.025, .975), df=5)
[1] -2.570582  2.570582
```

Se V_1 e V_2 são duas variáveis aleatórias independentes com distribuição chi quadrática com grau de liberdade m_1 e m_2 respectivamente, então a seguinte quantidade segue uma **distribuição F** com numerador m_1 grau de liberdade e denominador m_2 grau de liberdade (m_1,m_2).

$$F = \frac{V_1/m_1}{V_2/m_2} \sim F_{(m_1,m_2)}$$

Abaixo temos um gráfico de uma distribuição F com (5,2) graus de liberdade:



Encontre o percentil 95 de uma distribuição F com (5,2) graus de liberdade. Em R achamos assim:

```
> qf(.95, df1=5, df2=2)
[1] 19.29641
```

1.12.4 - Estimativas de intervalo

A estimativa pontual da média de uma população (exemplo survey de library MASS) pode ser feita facilmente em R com:

```
> library(MASS)
> altura<-survey$Height
> mean(altura, na.rm=TRUE)
[1] 172.3809
```

Agora vamos estimar a média de uma população com variância conhecida para estimarmos sua precisão.

Assumindo que o desvio padrão da população 'survey' é 9.48, encontre a margem de erro e estimativa de intervalo para um nível de confiança de 95%. Em R executamos:

```
> altura<-na.omit(survey$Height) #se livra dos NA
> n = length(altura)
> sigma<-9.48
> sem<-sigma/sqrt(n) #desvio padrão da média
> sem
[1] 0.6557453
> E<-qnorm(0.975)*sem
```

```
> E  #margem de erro
[1] 1.285237
> xbar<-mean(altura)
> xbar +c(-E, E)
[1] 171.0956 173.6661
```

E agora para o caso de desconhecermos a variância

Sem assumir o desvio padrão da população altura dos estudantes, encontraremos a margem de erro com 95% de confiança usando R assim:

```
> altura<-na.omit(survey$Height) #se livra dos NA
> n = length(altura)
> s<-sd(altura)
> SE<- s/sqrt(n)
> SE
[1] 0.6811677
> E<-qt(0.975,df=n-1)*SE
> E
[1] 1.342878
> xbar<-mean(altura)
> xbar +c(-E, E)
[1] 171.0380 173.7237
```

A qualidade de uma amostragem pode ser melhorada pelo aumento do tamanho da amostragem.

Assumindo um desvio padrão de 9.48 qual seria o tamanho da amostragem para atingir uma margem de erro de 1.2 centímetros com nível de confiança de 95%? Usando R chegamos a 240:

```
> z<-qnorm(0.975)
> sigma<- 9.48
> E<-1.2
> z^2*sigma^2/E^2
[1] 239.7454
```

É possível estimar a proporção de uma característica populacional com base numa amostragem. Vamos encontrar agora a proporção de estudantes mulheres numa universidade com base na amostragem survey:

```
> library(MASS)
> sexo<-na.omit(survey$Sex)
> n<-length(sexo)
> k <-sum(sexo =="Female")
> pbar<- k/n
> pbar
[1] 0.5
```

O resultado foi 50%.

E agora para estimar o intervalo de uma proporção populacional usamos com 95% de nível de confiança:

```
> SE<-sqrt(pbar*(1-pbar)/n) #Erro padrão
> SE
[1] 0.03254723
> E<- qnorm(0.975)*SE
> E
[1] 0.06379139
> pbar + c(-E, E)
[1] 0.4362086 0.5637914
```

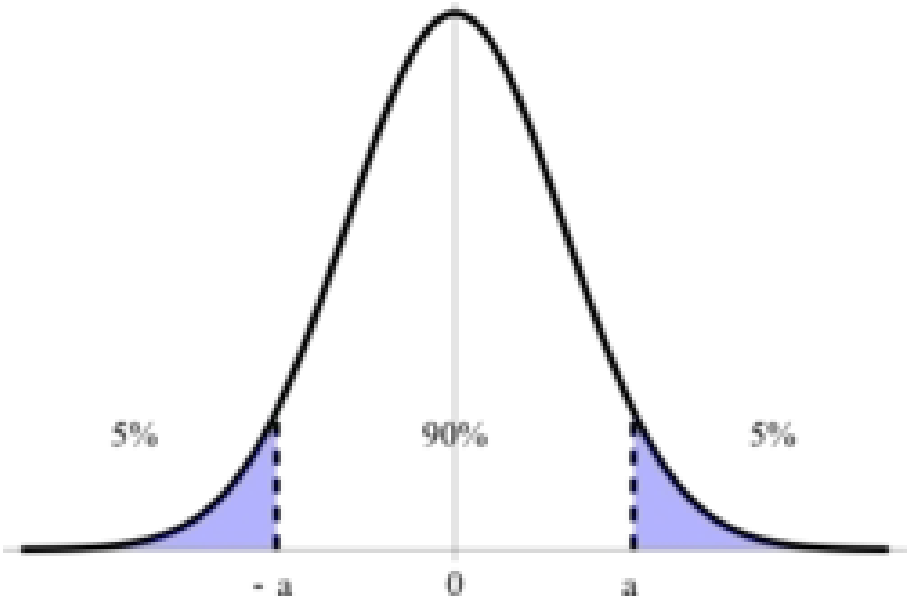
A 95% de nível de confiança entre 43.6% e 56.3% dos estudantes da universidade são mulheres.

Finalizando vamos calcular o tamanho da amostragem necessária para termos uma margem de erro de 5% num nível de confiança de 95%:

```
> z<-qnorm(0.975)
> p<-0.5
> E<-0.05
> z^2*p*(1-p)/E^2
[1] 384.1459
```

Precisaríamos 385 amostras para garantir margem de erro de 5% a um nível de confiança de 95% .

1.12.5 - Testando hipóteses



Cauda inferior (esquerda) e superior (direita) em cinza

A hipótese nula de um **teste de cauda inferior da média de população** pode ser expressa por:

$$\mu \geq \mu_0$$

onde μ_0 o limite inferior hipotético da média μ de uma população verdadeira.

Vamos definir o teste estatístico z em termos da média, do tamanho da amostra e do desvio padrão σ.

$$z = \frac{\bar{x} - \mu_0}{\sigma/\sqrt{n}}$$

Então a hipótese nula do teste da cauda inferior é para ser rejeitada se $z<=-z_\alpha$ onde z_α é o percentil 100(1-α) de uma distribuição normal padrão.

Ex.: Foi dito que o teor médio de um jazimento é acima de 10%. Em uma amostragem de 30 pontos deste jazimento chegamos a um valor médio de 9.9%. Assumindo um desvio padrão de 0.12%. Num nível de significância de 0.05, podemos rejeitar o teor anunciado? Vamos usar R nesse teste:

```
> teor<- 9.9     #teor medio das 30 amostras
> mi0<-  10      #teor a ser aceito ou rejeitado
> sigma<- 0.12   #desvio padrão
> n<- 30         #tamanho da amostra
> z<- (teor-mi0)/(sigma/sqrt(n))
> z             #teste estatístico
[1] -4.564355
> alfa<- 0.05
> z.alfa<- qnorm(1-alfa)
> -z.alfa       #valor crítico
[1] -1.644854
```

Como $z<=-z_\alpha$ a hipótese de o teor ser >= 10% está rejeitada a nível de significância de 0.05 .

Da mesma forma, a hipótese nula de um **teste de cauda superior da média de população** pode ser expressa por:

$$\mu \leq \mu_0$$

onde μ_0 o limite inferior hipotético da média μ de uma população verdadeira.

Então a hipótese nula do teste da cauda superior é para ser rejeitada se $z>=z_\alpha$ onde z_α é o percentil 100(1-α) de uma distribuição normal padrão.

Ex: Em uma frente de lavra a porcentagem de determinado contaminante de minério foi estipulada em ser no máximo 2%. Em uma amostra de 35 pontos encontramos uma média de 2.1% deste contaminante, assumindo um desvio padrão de 0.25% da população. A 0.05 de nível de significância, podemos rejeitar o valor de no máximo 2% de contaminante?

```
> teor<-2.1    #teor médio das 35 amostras
> mi0<- 2      #teor máximo a ser aceito ou rejeitado
> sigma<-0.25  #desvio padrão
> n=35         #tamanho da amostra
> z<-(teor-mi0)/(sigma/sqrt(n))
> z            #teste estatístico
[1] 2.366432
> alfa<-0.05
> z.alfa<-qnorm(1-alfa)
> z.alfa      #valor crítico
[1] 1.644854
```

Como $z>=z_\alpha$ a hipótese de o teor ser <= 2% está rejeitada a nível de significância de 0.05 .

Agora veremos a hipótese nula de um **teste das duas caudas da média de população** pode ser expressa por:

$$\mu = \mu_0$$

onde μ_0 o limite inferior hipotético da média μ de uma população verdadeira.

Então a hipótese nula do teste das duas caudas é para ser rejeitada se $z>=z_{\alpha/2}$ ou $z<=-z_{\alpha/2}$ onde $z_{\alpha/2}$ é o percentil 100(1-α/2) de uma distribuição normal padrão.

Ex.: Numa mina de ouro foi definido um teor médio de 15.4 g/ton de ouro. Abrindo uma nova frente de lavra obtivemos 35 amostras que geraram um teor médio de 14.6 g/ton de ouro, Sabemos

que o desvio padrão da reserva é de 2.5 g/ton. A 0.05 de nível de significância podemos rejeitar a hipótese de que o teor encontrado não difere do teor da reserva?

```
> teor<-14.6    #teor médio das 35 amostras
> mi0<- 15.4    #teor a ser aceito ou rejeitado
> sigma<-2.5    #desvio padrão
> n=35          #tamanho da amostra
> z<-(teor-mi0)/(sigma/sqrt(n))
> z             #teste estatístico
[1] -1.893146
> alpha<-0.05
> z.meio_alfa<-qnorm(1-alfa/2)
> c(-z.meio_alfa, z.meio_alfa) # faixa crítica
[1] -1.959964  1.959964
```

Como z está entre z_α e $-z_\alpha$ não podemos rejeitar a hipótese e o teor encontrado na nova frente é válido.

Caso não saibamos a variância da população original podemos usar os testes acima ***usando o desvio padrão da sua amostra ao invés do da população*** e usar Student T (função qt(1-alfa, df=n-1)) no lugar de Normal (função qnorm(1-alfa)) para determinar a faixa ou valor crítico do teste.

A hipótese nula do **teste de cauda inferior da proporção populacional** pode ser expressa por:

$$p \geq p_0$$

onde p_0 seria o limite inferior hipotético da proporção populacional p.

Vamos definir o teste estatístico z em termos da proporção populacional e do tamanho da amostra.

$$z = \frac{\bar{p} - p_0}{\sqrt{p_0(1-p_0)/n}}$$

Então a hipótese nula do teste da cauda inferior é para ser rejeitada se $z<=-z_\alpha$ onde z_α é o percentil 100(1-α) de uma distribuição normal padrão.

Ex.: Em uma campanha de exploração mineral tivemos 60% de furos bem-sucedidos no ano passado. Aleatoriamente escolhemos 148 furos feitos esse ano e 85 foram positivos. A um nível de significância de 0.05 podemos rejeitar que atingimos mais de 60% de sucesso esse ano?

```
> pp<- 85/148    #proporção da amostra
> p0<- 0.6       # valor hipotético
> n<- 148        #tamanho da amostra
> z<- (pp-p0)/sqrt(p0*(1-p0)/n)
> z              # teste estatístico
[1] -0.6375983
> alfa<- 0.05
> z.alfa<-qnorm(1-alfa)
> -z.alfa       # valor crítico
[1] -1.644854
```

O valor de z não é menor que o de z_0, assim não rejeitamos a hipótese e o sucesso nos furos está acima de 60% este ano usando um nível de significância de 0.05.

A hipótese nula do **teste de cauda superior da proporção populacional** pode ser expressa por:

$$p \leq p_0$$

onde p_0 seria o limite inferior hipotético da proporção populacional p.

Então a hipótese nula do teste da cauda inferior é para ser rejeitada se $z >= z_\alpha$ onde z_α é o percentil $100(1-\alpha)$ de uma distribuição normal padrão.

Ex.: Numa sondagem de delineação de um depósito de carvão a perda de testemunho não deve ser maior que 12% dos furos. Foram selecionados aleatoriamente 214 furos e nestes observou-se perda em 30 furos. Para um nível de 0.05 podemos rejeitar termos menos de 12% de perda?

```
> pp<- 30/214    #proporção da amostra
> p0<- 0.12       # valor hipotético
> n<- 214        #tamanho da amostra
> z<- (pp-p0)/sqrt(p0*(1-p0)/n)
> z              # teste estatístico
[1] 0.908751
> alfa<- 0.05
> z.alfa<-qnorm(1-alfa)
> z.alfa       # valor crítico
[1] 1.644854
```

Z não é maior que z_0 , não rejeitamos a hipótese e assim a perda de amostra em furos é menor que 12%.

A hipótese nula do **teste das duas caudas da proporção populacional** pode ser expressa por:

$$p = p_0$$

onde p_0 seria o limite inferior hipotético da proporção populacional p.

Então a hipótese nula do teste das duas caudas é para ser rejeitada se $z <= -z_\alpha$ ou $z >= z_C$ onde z_α é o percentil $100(1-\alpha)$ de uma distribuição normal padrão.

Ex.: Jogando cara ou coroa 20 vezes obtivemos 12 caras. Com um nível de significância de 0.05 pode-se rejeitar a hipótese de que o cara e coroa é justo?

```
> pp<- 12/20    #proporção da amostra
> p0<- 0.5       # valor hipotético
> n<- 20        #tamanho da amostra
> z<- (pp-p0)/sqrt(p0*(1-p0)/n)
> z              # teste estatístico
[1] 0.8944272
> alfa<- 0.05
> z.meio_alfa<-qnorm(1-alfa/2)
> c(-z.meio_alpha, z.meio_alpha)
[1] -1.959964  1.959964
```

O resultado está entre $-z_\alpha$ e z_α então não rejeitamos a hipótese de que o cara e coroa é justo a um nível de significância de 0.05.

1.12.6 - Erro do tipo II

Num **teste de cauda inferior da média de população** a hipótese nula diz que a média da população μ é maior que um valor hipotético μ_0 .

$$\mu \geq \mu_0$$

Um **erro tipo II** ocorre se o teste hipótese, baseado em uma amostragem aleatória, falha em ser rejeitado mesmo quando a verdadeira média da população μ é em fato menor que μ_0.

Assumimos que a população tem **variância σ^2 conhecida**. Pelo Teorema do Limite Central, a população de todas as médias das amostras possíveis quando aumentamos o tamanho da amostra n, se aproxima de uma distribuição normal. Assim nós podemos calcular a faixa de amostragem média para qual a hipótese nula não será rejeitada, e obter uma estimativa da probabilidade de erro do tipo II.

Ex.: Foi dito que o teor médio de um jazimento é acima de 10%. Assumindo um desvio padrão de 0.12% e um teor médio atual de 9.95%. Num nível de significância de 0.05, qual seria a probabilidade de termos um erro tipo II para uma amostragem aleatória de 30 pontos?

```
> n<- 30                # tamanho da amostragem
> sigma<- 0.12          # desvio padrão da população
> sem<- sigma/sqrt(n)   # erro padrão
> sem
[1] 0.0219089
> alfa<- 0.05           # nível de significância
> mi0<- 10              #limite inferior hipotético
> q<- qnorm(alfa,mean=mi0,sd=sem)
> q
[1] 9.963963
> ma<- 9.95             # média assumida
> pnorm(q,mean=ma,sd=sem,lower.tail=FALSE)
[1] 0.261957
```

Se a amostragem de 30 pontos com média de teor 9.95% e com um desvio padrão da população dado de 0.12%, então a probabilidade de erro tipo II testando a hipótese nula μ>=10% a 0.05 significância é 26.2% e a força do teste da hipótese e 73.8%.

Num **teste de cauda superior da média de população** a hipótese nula diz que a média da população μ é maior que um valor hipotético μ_0 .

$$\mu \leq \mu_0$$

Um **erro tipo II** ocorre se o teste hipótese, baseado em uma amostragem aleatória, falha em ser rejeitado mesmo quando a verdadeira média da população μ é em fato maior que μ_0.

Ex.: Em uma frente de lavra a porcentagem de determinado contaminante de minério foi estipulada em ser no máximo 2%. Assumindo que a média atual é de 2.09% deste contaminante e assumindo um desvio padrão da população original de 0.25%. A 0.05 de nível de significância qual é a probabilidade de termos erro Tipo II para uma amostra de 35 pontos?

```
> n<- 35                # tamanho da amostragem
> sigma<- 0.25          # desvio padrão da população
> sem<-  sigma/sqrt(n)  # erro padrão
> sem
[1] 0.04225771
> alfa<- 0.05           # nível de significância
> mi0<- 2                # limite superior hipotético
> q<- qnorm(alfa,mean=mi0,sd=sem,lower.tail=FALSE)
```

```
> q
[1] 2.069508
> ma<- 2.09              # média assumida
> pnorm(q,mean=ma,sd=sem)
[1] 0.3138612
```

Para uma amostra de 35 pontos com média de 2.09% de contaminante e com desvio padrão da população original de 0.25%, a probabilidade de erro tipo II testando a hipótese nula μ<=2 a 0.05 de nível de significância é 31.4% e a força do teste de hipótese é de 68.6%.

Num **teste de duas caudas da média de população** a hipótese nula diz que a média da população μ é maior que um valor hipotético μ_0 .

$$\mu = \mu_0$$

Um **erro tipo II** ocorre se o teste hipótese, baseado em uma amostragem aleatória, falha em ser rejeitado mesmo quando a verdadeira média da população μ é em fato diferente de μ_0.

Ex.: Numa mina de ouro foi definido um teor médio de 15.4 g/ton de ouro. Abrindo uma nova frente de lavra amostramos um teor médio de 14.6 g/ton de ouro, Sabemos que o desvio padrão da reserva é de 2.5 g/ton. A 0.05 de nível de significância qual a probabilidade de termos um erro tipo II para um tamanho de amostra de 35?

```
> n<- 35                 # tamanho da amostragem
> sigma<- 2.5            # desvio padrão da população
> sem<- sigma/sqrt(n)    # erro padrão
> sem
[1] 0.4225771
> alfa<- 0.05           # nível de significância
> mi0<- 15.4            # média hipotética
> I <- c(alfa/2, 1-alfa/2)
> q=qnorm(I,mean=mi0,sd=sem)
> q
[1] 14.57176 16.22824
> ma<- 14.6             # média assumida
> p<- pnorm(q,mean=ma,sd=sem)
> p
[1] 0.4733631 0.9999417
> diff(p)             # Diferença entre os dois pontos extremos
[1] 0.5265786
```

Para 35 amostras com média 14.6 g/ton e desvio padrão da população original de 2.5 g/ton a probabilidade de erro tipo II testando a hipótese nula μ=14.6 g/ton a um nível de significância de 0.05 é 52.6% e a força do teste de hipótese é 47.4%.

Quando não temos informação sobre a variância da população **podemos chegar à probabilidade de erro tipo II usando o desvio padrão da amostra** e usar distribuição Student T com n-1 grau de liberdade conforme:

$$\frac{\bar{x} - \mu}{s/\sqrt{n}}$$

Isso nos permitirá calcular a faixa das médias das amostras para as quais a hipótese nula não será rejeitada e obter a probabilidade do erro tipo II conforme mostraremos abaixo:

Teste de cauda inferior da média de população (variância desconhecida)

Foi dito que o teor médio de um jazimento é acima de 10%. Numa amostragem de 30 pontos obtivemos um desvio padrão de 0.125% e um teor médio atual de 9.95%. Num nível de significância de 0.05, qual seria a probabilidade de termos um erro tipo II?

```
> n <-  30                # tamanho da amostra
> s <-  0.125             # desvio padrão da amostra
> SE <-  s/sqrt(n)        # estimativa do erro padrão
> SE
[1] 0.02282177
> alfa <-  0.05           # nível de significância
> mi0 <-  10              # limite inferior hipotético
> q <-  mi0+qt(alfa,df=n-1)*SE
> q
[1] 9.961223
> mi <-  9.95             # média assumida atual
> pt((q-mi)/SE,df=(n-1),lower.tail=FALSE)
[1] 0.3132943
```

Neste caso a probabilidade do erro tipo II testando a hipótese nula μ>=10 a 0.05 nível de significância é de 31.3% e a força do teste de hipótese é de 68.7%.

Teste de cauda superior da média de população (variância desconhecida)

Em uma frente de lavra a porcentagem de determinado contaminante de minério foi estipulada em ser no máximo 2%. Numa amostragem de 35 pontos a média atual é de 2.09% deste contaminante e temos na amostra um desvio padrão de 0.3%. A 0.05 de nível de significância qual seria a probabilidade de termos um erro tipo II?

```
> n <- 35                #tamanho da amostra
> s <- 0.3               # desvio padrão da amostra
> SE <- s/sqrt(n)        # estimativa do erro padrão
> SE
[1] 0.05070926
> alfa <- 0.05          # nível de significância
> mi0 <- 2              # limite superior hipotético
> q <- mi0+qt(alfa,df=n-1,lower.tail=FALSE)*SE
> q
[1] 2.085746
> mi <- 2.09             # média assumida atual
> pt((q-mi)/SE,df=(n-1))
[1] 0.4668141
```

Neste caso a probabilidade do erro tipo II testando a hipótese nula μ<=2 a 0.05 nível de significância é de 46.7% e a força do teste de hipótese é de 53.3%.

Teste das duas caudas da média de população (variância desconhecida)

Numa mina de ouro foi definido um teor médio de 15.4 g/ton de ouro. Abrindo uma nova frente de lavra em 35 amostras um teor médio de 14.6 g/ton de ouro e desvio padrão da amostra é de 2.5 g/ton. A 0.05 de nível de significância qual seria a probabilidade de termos um erro tipo II?

```
> n <-  35                #tamanho da amostra
> s <-  2.5               # desvio padrão da amostra
> SE <-  s/sqrt(n)        # estimativa do erro padrão
> SE
[1] 0.4225771
> alfa <-  0.05             # nível de significância
> mi0 <-  15.4              # média hipotética
```

```
> I <-  c(alfa/2,1-alfa/2)
> q <-  mi0+qt(I,df=n-1)*SE
> q
[1] 14.54122 16.25878
> mi <-  14.6
> p <-  pt((q-mi)/SE,df=n-1)
> p
[1] 0.4450963 0.9997996
> diff(p)         #diferença dos extremos
[1] 0.5547033
```

Neste caso a probabilidade do erro tipo II testando a hipótese nula μ=15.4 a 0.05 nível de significância é de 55.5% e a força do teste de hipótese é de 44.5%.

1.12.7 - Interferência entre duas populações

Duas amostras de dados são pareadas se elas vem de repetidas observações de um mesmo indivíduo. Aqui nós assumimos que o dado das populações obedecem uma distribuição normal. Usando o teste T pareado podemos obter uma estimativa de intervalo da diferença das médias das populações.

Ex.: s1 e s31 são imagens raster da banda 2 (sentinel2) de um mesmo local adquiridas em dias diferentes (dia 01/07/2018 e dia 31/07/2018). Extraímos os primeiros 10000 valores de cada raster e efetuamos o teste T pareado:

```
> xs1<-extract(s1,c(1:10000))
> xs31<-extract(s31,c(1:10000))
> t.test(xs1,xs31,paired=TRUE)

        Paired t-test

data:  xs1 and xs31
t = 36.935, df = 9999, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 101.0702 112.3996
sample estimates:
mean of the differences
               106.7349
```

O intervalo com 95% confiança da diferença na média das imagens dos dias 1 e 31 é o intervalo entre 101.0700 e 112.3996.

Duas amostras de dados são consideradas independentes se elas vem de populações não relacionáveis e as amostras não afetam cada uma delas. Aqui nós assumimos que o dado das populações obedecem uma distribuição normal. Usando o teste T não pareado nós podemos obter uma estimativa de intervalo da diferença entre as medias das duas populações.

Ex.: Suponhamos que temos dois depósitos minerais 0 e 1 em diferentes áreas e resultados de vários furos dos dois depósitos são apresentados conforme abaixo:

Furo	Espessura(m)	Teor/metro (%/m)	Depósito
BZ2	4.5	15	0
BZ3	4.1	14.8	0
BZ4	3.9	15.7	0
BZ5	5.5	12.6	0

BZ6	4.3	14.5	0
BZ7	2.9	17.4	0
BZ8	3.5	14.6	0
BZ9	4.0	12.5	0
BZ1	5.6	11.4	0
BZ10	3.4	15.7	0
AT2	2.2	22.0	1
AT3	2.2	28.3	1
AT4	1.8	26.5	1
AT5	3.6	20.7	1
AT6	2.2	22.3	1
AT7	1.5	25.4	1
AT8	2.7	22.4	1
AT9	2.2	24.3	1
AT1	3.5	22.2	1
AT10	1.2	23.1	1

Assumindo que os resultados seguem uma distribuição normal, encontre a estimativa do intervalo, 95% de confiança, da diferença entre a média dos teores entre os dois depósitos:

```
> depo$Depósito
 [1] 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
> depo$Teor
 [1] 15.0 14.8 15.7 12.6 14.5 17.4 14.6 12.5 11.4 15.7 22.0 28.3 26.5 20.7 22.3
[16] 25.4 22.4 24.3 22.2 23.1
> D<-depo$Depósito == 0
> teor.0<-depo[D,]$Teor      # teor do deposito 0
> teor.1<-depo[!D,]$Teor     # teor do deposito 1
> teor.0
 [1] 15.0 14.8 15.7 12.6 14.5 17.4 14.6 12.5 11.4 15.7
> teor.1
 [1] 22.0 28.3 26.5 20.7 22.3 25.4 22.4 24.3 22.2 23.1
> t.test(teor.0,teor.1)

        Welch Two Sample t-test

data:  teor.0 and teor.1
t = -9.9143, df = 16.754, p-value = 2.015e-08
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -11.281302  -7.318698
sample estimates:
mean of x mean of y
    14.42     23.72
```

Nestes resultados, a média do depósito 0 é 14.43%/m e a média do depósito 1 é 23.72%/m. O intervalo da diferença das médias de teores variam entre 7.32%/m e 11.28%/m, a 95% confiança.

Um levantamento em duas populações distintas produzirá resultados diferentes. É frequentemente necessário comparar a proporção da resposta do levantamento entre as duas populações. Aqui assumimos que os dados das populações seguem uma distribuição normal.

Ex.:Crianças de uma cidade australiana são classificadas por etnia (A,N), gênero(M,F), e outras informações. Assumindo que os dados seguem a uma distribuição normal, a 95% de confiança qual é a proporção de meninas entre as populações aborígenes(A) e não aborígenes(N).

```
> library(MASS)
> table(quine$Eth, quine$Sex)
    F  M
  A 38 31
  N 42 35
```

```
> prop.test(table(quine$Eth, quine$Sex), correct=FALSE)

        2-sample test for equality of proportions without continuity
        correction

data:  table(quine$Eth, quine$Sex)
X-squared = 0.0040803, df = 1, p-value = 0.9491
alternative hypothesis: two.sided
95 percent confidence interval:
 -0.1564218  0.1669620
sample estimates:
   prop 1    prop 2
0.5507246 0.5454545
```

A proporção de meninas entre as populações aborígenes(A) e não aborígenes(N), a 95% confiança, está entre -15.6% e 16.7%.

1.12.8 - Testes de adequação (aderência)

Uma população é chamada multinomial se seus dados forem em categorias e pertencerem a uma coleção de classes discretas não sobrepostas.

A hipótese nula do **teste de aderência para distribuição multinomial** é que a frequência observada f, é igual a uma contagem esperada e, em cada categoria. Ela será rejeitada se 0 valor-p do teste chi quadrado é menor que o valor de significância fornecido α.

$$\chi^2 = \sum_i \frac{(f_i - e_i)^2}{e_i}$$

Ex.: No levantamento survey temos a coluna Smoke(fumante) com as opções de resposta, “Heavy” (pesado),”Regul” (regular), “Occas” (ocasional) e “Never” (nunca). Este dado é multinomial. Supondo uma estatística no campus mostrada abaixo, demonstre se o dado em survey confirma estatística em um nível de significância de 0.05.

Heavy Never Occas Regul
4.5% 79.5% 8.5% 7.5%

```
> library(MASS)
> levels(survey$Smoke)
[1] "Heavy" "Never" "Occas" "Regul"
> smoke.freq<-table(survey$Smoke)
> smoke.freq                          # resultado da tabela survey
Heavy Never Occas Regul
   11   189    19    17
> smoke.prob<-c(0.045,0.795,0.085,0.075) #dados da estatística
> chisq.test(smoke.freq,p=smoke.prob)    #test chi-quadrado

        Chi-squared test for given probabilities

data:  smoke.freq
X-squared = 0.10744, df = 3, p-value = 0.9909
```

Como o valor-p 0.9909 é maior que o valor de significância 0.05 nós não rejeitamos a hipótese nula de que os dados de survey suportam os resultados da estatística no campus.

Duas variáveis x e y são chamadas **independentes** se a distribuição de probabilidade de uma variável não é afetada pela presença da outra.

Assumindo que f_Y é a contagem da frequência observada dos eventos pertencentes a ambas categorias i-ésima de x e j-ésima de y. Também assumindo que e_Y ser a contagem correspondente esperada de x e y e são independentes. A hipótese nula do **teste de independência** será rejeitada se o valor-p do seguinte teste chi quadrado é menor que o valor de significância fornecido α.

$$\chi^2 = \sum_{i,\ j} \frac{(f_{ij} - e_{ij})^2}{e_{ij}}$$

Ex.: Vamos agora testar a independência usando os dados survey checando o hábito de fumar com o hábito de se exercitar em nível de significância de 0.05.

```
> library(MASS)
> tbl <- table(survey$Smoke, survey$Exer)
> tbl

       Freq None Some
  Heavy    7    1    3
  Never   87   18   84
  Occas   12    3    4
  Regul    9    1    7
>  ctbl<-cbind(tbl[,"Freq"],tbl[,"None"]+tbl[,"Some"])  #combinamos  os  valores
pequenos de None e Some
> ctbl
      [,1] [,2]
Heavy    7    4
Never   87  102
Occas   12    7
Regul    9    8
> chisq.test(ctbl)

        Pearson's Chi-squared test

data:  ctbl
X-squared = 3.2328, df = 3, p-value = 0.3571
```

Como o valor-p de 0.357 é maior que 0.05 de significância, nós não rejeitamos a hipótese de que os hábitos de fumar e se exercitar são independentes.

1.12.9 - Análise de Variância (ANOVA)

Análise da variância (ANOVA) é dependente do design.

No **design completamente aleatório**, existe somente um fator primário a ser considerado no experimento. Os indivíduos testados são designados para níveis de tratamento do fator primário aleatoriamente.

Ex.: Três elementos de um depósito são testados. 18 pontos de amostragem foram escolhidos aleatoriamente. Para estar de acordo com o **design completamente aleatório,** os resultados do primeiro elemento vieram de 6 pontos aleatoriamente escolhidos, o segundo elemento de outros 6 e mesmo para o terceiro elemento. A uma nível de significância de 0.05, teste se a média do resultado para os 3 elementos são iguais.

Copie e salve como anov1.csv os dados abaixo:

```
el1, el2, el3
22,52,16
42,33,24
44,8,19
52,57,18
45,43,34
37,32,39
```

```
> df1<-read.csv('anov1.csv')
> df1
  el1 el2 el3
1  22  52  16
2  42  33  24
3  44   8  19
4  52  47  18
5  45  43  34
6  37  32  39
> r<-c(t(as.matrix(df1))) #concatenando df1 como um vetor
> r
 [1] 22 52 16 42 33 24 44  8 19 52 47 18 45 43 34 37 32 39
> # designando as variáveis para o tratamento
> f<- c('el1','el2','el3')    #níveis do tratamento
> k<-3                        # número de níveis de tratamento
> n<-6                        # observações por tratamento
> tm<-gl(k,1,n*k,factor(f))   # cria vetor corresp. aos níveis de tratam.
> tm
 [1] el1 el2 el3 el1 el2 el3 el1 el2 el3 el1 el2 el3 el1 el2 el3 el1 el2 el3
Levels: el1 el2 el3
> ANOVA<-aov(r ~ tm)
> summary(ANOVA)
            Df Sum Sq Mean Sq F value Pr(>F)
tm           2  745.4   372.7   2.541  0.112
Residuals   15 2200.2   146.7
```

Como o valor-p de 0.112 é maior que 0.05 de nível de significância, nós não rejeitamos a hipótese nula de que as médias dos elementos são iguais.

Em um **design aleatório em blocos**, existe somente um fator primário em consideração no experimento. Indivíduos similares do teste são agrupados em **blocos**. Cada bloco é testado contra todos os níveis de tratamento do fator primário em ordem aleatória com a intenção de eliminar possível influência de fatores externos.

Ex.: Três elementos de um depósito são testados. 6 pontos de amostragem foram escolhidos aleatoriamente. Para estar de acordo com o **design aleatório em blocos** cada ponto amostrará os três elementos. A um nível de significância de 0.05, teste se a média do resultado para os 3 elementos são iguais.

Copie e salve como anov2.csv os dados abaixo:

```
el1, el2, el3
31,27,24
31,28,31
45,29,46
21,18,48
42,36,46
32,17,40
```

```
> df1<-read.csv('anov2.csv')
> df1
  el1 el2 el3
1  31  27  24
2  31  28  31
3  45  29  46
4  21  18  48
5  42  36  46
6  32  17  40
> r<-c(t(as.matrix(df1))) #concatenando df1 como um vetor
> r
 [1] 31 27 24 31 28 31 45 29 46 21 18 48 42 36 46 32 17 40
> # designando as variáveis para o tratamento
> f<- c('el1','el2','el3')    #níveis do tratamento
> k<-3                        # número de níveis de tratamento
> n<-6                        # observações por tratamento
> tm<-gl(k,1,n*k,factor(f))   # cria vetor correp. aos níveis de tratam.
> tm
 [1] el1 el2 el3 el1 el2 el3 el1 el2 el3 el1 el2 el3 el1 el2 el3 el1 el2 el3
Levels: el1 el2 el3
> blk<-gl(n,k,k*n)           # fator bloco
> blk
 [1] 1 1 1 2 2 2 3 3 3 4 4 4 5 5 5 6 6 6
Levels: 1 2 3 4 5 6
> ANOVA<- aov(r ~ tm + blk)
> summary(ANOVA)
            Df Sum Sq Mean Sq F value Pr(>F)
tm           2  538.8  269.39   4.959 0.0319 *
blk          5  559.8  111.96   2.061 0.1547
Residuals   10  543.2   54.32
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
>
```

Como o valor-p de 0.032 é menor que o nível de significância 0.05, nós rejeitamos a hipótese nula de que as médias dos resultados dos elementos são iguais.

No **design fatorial** existem mais de um fator em consideração no experimento. Os indivíduos testados são designados para o tratamento de cada fator em combinações aleatórias.

Ex.: Três elementos de dois depósitos similares são testados. 12 pontos de amostragem foram escolhidos aleatoriamente em cada depósito. Para estar de acordo com o **design fatorial** os resultados dos elementos vieram de 4 pontos aleatoriamente escolhidos, o segundo elemento de outros 4 e mesmo para o terceiro elemento. A um nível de significância de 0.05, teste se a média do resultado para os 3 elementos são iguais e conclua se as médias entre os dois depósitos diferem.

Copie e salve como anov3.csv os dados abaixo:

```
el1,el2,el3
A1,25,39,36
A2,36,42,24
A3,31,39,28
A4,26,35,29
B1,51,43,42
B2,47,39,36
B3,47,53,32
B4,52,46,33
```

```
> df1<-read.csv('anov3.csv')
> df1
   el1 el2 el3
A1  25  39  36
A2  36  42  24
A3  31  39  28
A4  26  35  29
B1  51  43  42
B2  47  39  36
B3  47  53  32
B4  52  46  33
> r<-c(t(as.matrix(df1))) #concatenando df1 como um vetor
> r
 [1] 25 39 36 36 42 24 31 39 28 26 35 29 51 43 42 47 39 36 47 53 32 52 46 33
> # designando as variáveis para o tratamento
> f1<- c('el1','el2','el3')    #níveis do tratamento
> f2<- c('depA','depB')        #níveis do tratamento
> k1<-length(f1)               # número de níveis de tratamento fator 1
> k2<-length(f2)               # número fator 2
> n <-4
> tm1<-gl(k1,1,n*k1*k2,factor(f1))
> tm1
 [1] el1 el2 el3 el1 el2 el3 el1 el2 el3 el1 el2 el3 el1 el2 el3 el1 el2 el3
el1
[20] el2 el3 el1 el2 el3
Levels: el1 el2 el3
> tm2<-gl(k2,n*k1,n*k1*k2,factor(f2))
> tm2
 [1] depA depA depA depA depA depA depA depA depA depA depA depA depB depB depB
[16] depB depB depB depB depB depB depB depB depB depB
Levels: depA depB
> ANOVA<-aov(r ~ tm1*tm2)
> summary(ANOVA)
            Df Sum Sq Mean Sq F value   Pr(>F)
tm1          2  385.1   192.5   9.554  0.00149 **
tm2          1  715.0   715.0  35.481 1.23e-05 ***
tm1:tm2      2  234.1   117.0   5.808  0.01132 *
Residuals   18  362.8    20.2
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

Como o valor-p de 0.0015 é menor que o nível de significância 0.05, nós rejeitamos a hipótese nula de que as médias dos resultados dos elementos são iguais. Além do mais o valor de 1.23e-05 para a comparação depósito A e B é também bem menor que o nível de significância 0.05 mostrando que existe uma diferença geral entre os valores dos elementos.

1.12.10 - Regressão linear simples

Se escolhermos os parâmetros α e β no **modelo de regressão linear simples** para minimizar o erro das soma dos quadrados ε, nós teremos então de ter a **equação estimada de regressão simples**. Ela nos permitirá calcular valores ajustados de y em função do valor x:

$$\hat{y} = a + bx$$

Ex.: Vamos usar a base de dados ‘faithful’ e estimar a duração da próxima erupção com base num tempo de espera de 80 minutos:

```
> erupção.ml<-lm(eruptions ~ waiting, data=faithful)
> coeficiente<- coefficients(erupção.ml)
> coeficiente
(Intercept)     waiting
-1.87401599  0.07562795
> #agora vamos ajustar o duração da erupção usando a equação de regressão
> espera<-80            # o tempo de espera
> duração<- coeficiente[1] + coeficiente[2]*espera
> duração
(Intercept)
    4.17622
```

Com base no modelo de regressão linear simples, se o tempo de espera for de 80 minutos nós esperaremos que a próxima erupção dure 4.17622 minutos.

O **coeficiente de determinação** de um modelo de regressão linear é o quociente das variâncias dos valores ajustados e valores observados de uma variável dependente. Se denotarmos y_i como os valores observados de uma variável dependente, $\bar{y}$ como sua média e $\hat{y}$ como o valor ajustado, então o coeficiente de determinação será:

$$r^2 = \frac{\sum(\hat{y}_i - \bar{y})^2}{\sum(y_i - \bar{y})^2}$$

Ex.: Encontre o **coeficiente de determinação** para a base de dados 'faithful':

```
> erupção.ml<-lm(eruptions ~ waiting, data=faithful)
> summary(erupção.ml)$r.squared
[1] 0.8114608
```

O coeficiente de determinação de 'faithful' é 0.8114608.

Assumindo que o erro ε no modelo de regressão linear é independente de x, e é normalmente distribuído com média zero e variância constante. Nós podemos definir se existe qualquer **relação de significância** entre x e y testando a hipótese nula que β=0.

```
> erupção.ml<-lm(eruptions ~ waiting, data=faithful)
> summary(erupção.ml)

Call:
lm(formula = eruptions ~ waiting, data = faithful)

Residuals:
     Min       1Q   Median       3Q      Max
-1.29917 -0.37689  0.03508  0.34909  1.19329

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.874016   0.160143  -11.70   <2e-16 ***
waiting      0.075628   0.002219   34.09   <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.4965 on 270 degrees of freedom
Multiple R-squared:  0.8115,    Adjusted R-squared:  0.8108
F-statistic:  1162 on 1 and 270 DF,  p-value: < 2.2e-16
```

Como o valor-p é muito menor que 0.05, nós rejeitamos a hipótese nula que β=0. Assim existindo uma relação de significância entre as variáveis no modelo de regressão linear da base de dados 'faithful'.

Assumindo que o erro ε no modelo de regressão linear é independente de x, e é normalmente distribuído com média zero e variância constante. Para um dado valor x, a estimativa do intervalo para a média da variável dependente, y, é chamado de **intervalo de confiança.**

Ex.: Para a base de dados ‘faithful’ , determine **intervalo com 95% confiança** para a media da duração da erupção para uma espera de 80 minutos.

```
> attach(faithful)
> erupção.ml<-lm(eruptions ~ waiting)
> novodado<- data.frame(waiting=80)
> predict(erupção.ml,novodado, interval="confidence")
       fit      lwr      upr
1 4.17622 4.104848 4.247592
> detach(faithful)
```

Para 95% intervalo de confiança, a média da duração da erupção para um tempo de espera de 80 minutos está entre 4.1048 e 4.2476.

Assumindo que o erro ε no modelo de regressão linear é independente de x, e é normalmente distribuído com média zero e variância constante. Para um dado valor x, a estimativa do intervalo da variável dependente, y, é chamado de **intervalo de predição.**

Ex.: Para a base de dados ‘faithful’ , determine **intervalo com 95% predição** da duração da erupção para uma espera de 80 minutos.

```
> attach(faithful)
> erupção.ml<-lm(eruptions ~ waiting)
> novodado<- data.frame(waiting=80)
> predict(erupção.ml,novodado, interval="predict")
       fit      lwr      upr
1 4.17622 3.196089 5.156351
> detach(faithful)
```

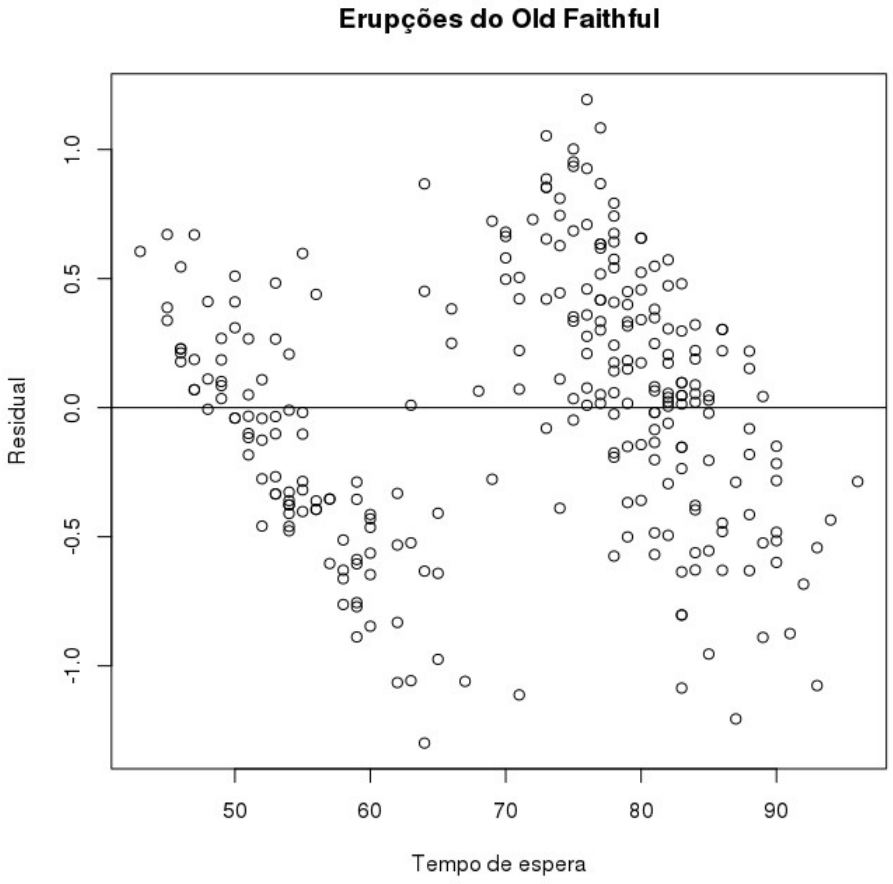
Para 95% intervalo de confiança, a duração da erupção para um tempo de espera de 80 minutos está entre 3.1961 e 5.1564.

O dado **residual** de um modelo de regressão linear simples é a diferença entre o dado observado da variável dependente y e o valor ajustado ŷ.

$$Residual = y - \hat{y}$$

Ex,: Plote o residual do modelo de regressão linear simples da base de dados ‘faithful’ contra a variável dependente ‘waiting’ (espera).

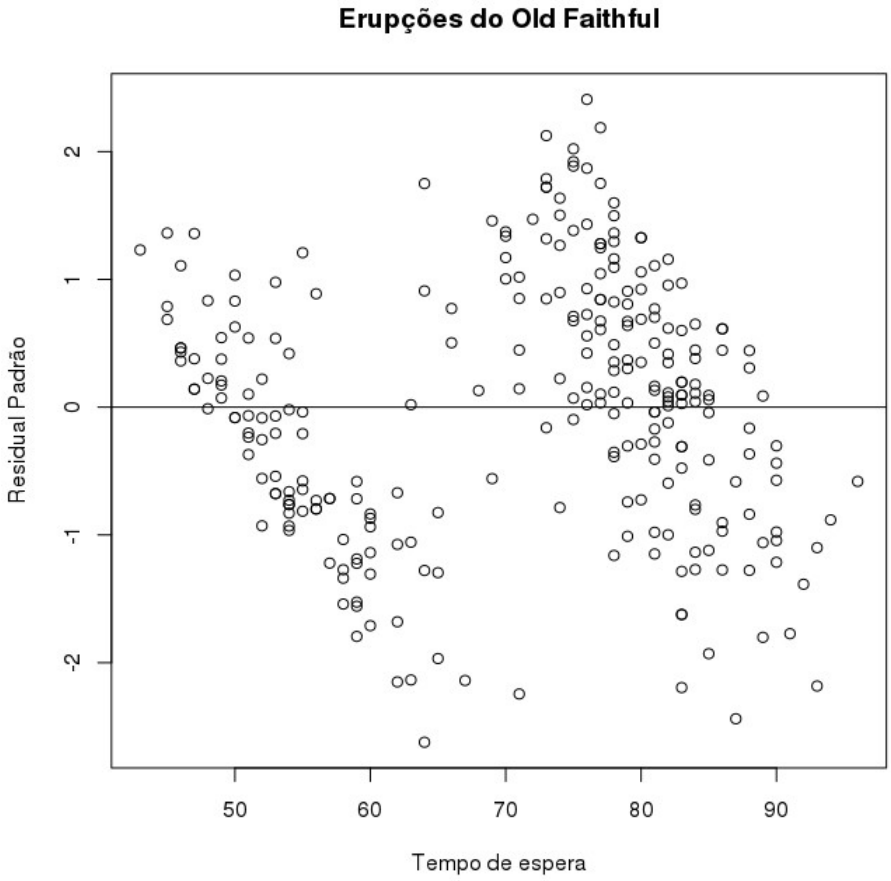
```
> erupção.ml<-lm(eruptions ~ waiting, data=faithful)
> erupção.residual<-resid(erupção.ml)
> plot(faithful$waiting,erupção.residual,
+ ylab='Residual',xlab='Tempo de espera',
+ main='Erupções do Old Faithful')
> abline(0,0)
```



O **residual padrão** é o residual dividido por seu desvio padrão.

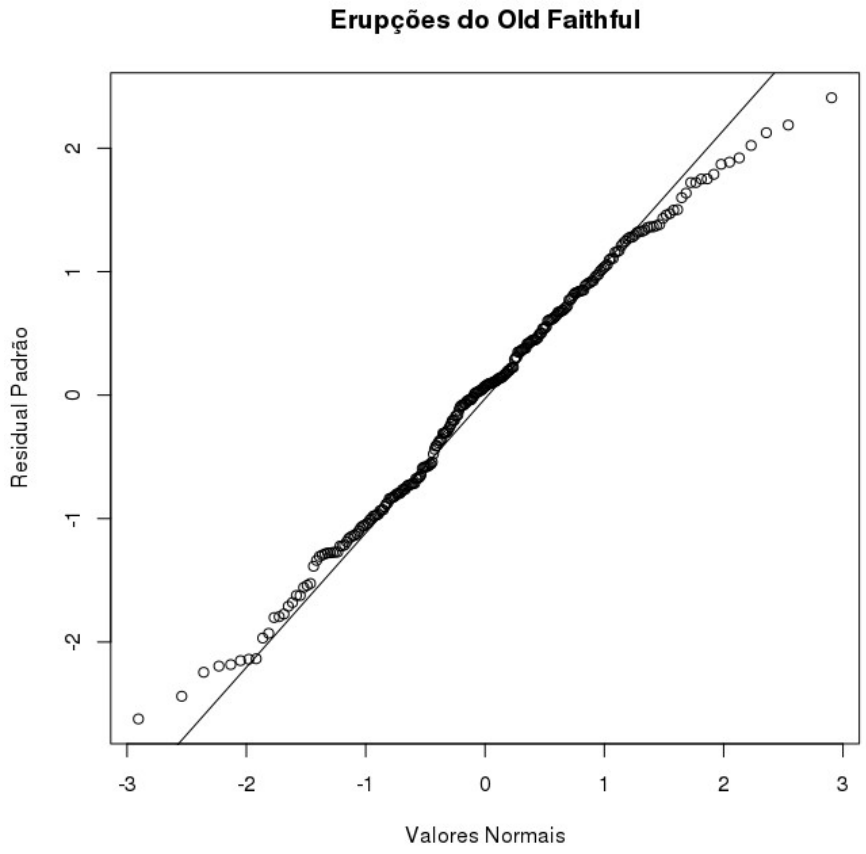
$$Standardized\ Residual\ i = \frac{Residual\ i}{Standard\ Deviation\ of\ Residual\ i}$$

```
> erupção.ml<-lm(eruptions ~ waiting, data=faithful)
> erupção.respadrão<-rstandard(erupção.ml)
> plot(faithful$waiting,erupção.respadrão,
+ ylab='Residual Padrão',xlab='Tempo de espera',
+ main='Erupções do Old Faithful')
> abline(0,0)
```



O **gráfico de probabilidade normal** é uma ferramenta gráfica para comparar uma base de dados com a distribuição normal. Nós podemos usar ele com o residual padrão do modelo de regressão linear e ver se o erro ε está de fato distribuído normalmente.

```
> erupção.ml<-lm(eruptions ~ waiting, data=faithful)
> erupção.respadrão<-rstandard(erupção.ml)
> qqnorm(erupção.respadrão,
+ ylab='Residual Padrão',xlab='Valores Normais',
+ main='Erupções do Old Faithful')
> qqline(erupção.respadrão)
```



Concluímos assim nossa revisão de estatística e como usamos ela com R.

1.13 – Informações finais

Na maioria dos casos se fará necessário instalar e usar um pacote de funções (package) adicional para trabalharmos em R.

Para instalar um pacote usamos (fazemos isso uma vez só)

```
> install.packages('nome_do_pacote')
```

Onde nome do pacote é substituído pelo nome real do pacote:

```
> install.packages('raster')
```

Carregamos o pacote instalado assim:

```
> library(raster)
```

Apenas demos uma pincelada no que diz respeito a R mas que permitirá a você desenvolver o seu conhecimento.

Pacotes usados nesta apostila:

→ **install.packages('raster')**
→ **install.packages('sp')**
→ **install.packages('rgdal')**
→ **install.packages('gstat')**
→ **install.packages('rgeos')**
→ **install.packages('plot3D')**
→ **install.packages('data.table')**
→ **install.packages('dismo')**
→ **install.packages('RColorBrewer')**
→ **install.packages('vioplot')**
→ **install.packages('tmap')**
→ **install.packages('spdep')**
→ **install.packages('spgwr')**
→ **install.packages('float')**
→ **install.packages('RPostgreSQL')**
→ **install.packages('rpostgis')**
→ **install.packages('cluster')**
→ **install.packages('randomForest')**
→ **install.packages('reshape2')**
→ **install.packages('ggplot2')**

PARTE 2 Manipulação de dados Espaciais

2.1 – Dados espaciais

Podemos classificar dados espaciais em duas categorias: discreta (um rio, uma cidade , um estado) e contínua (elevação, temperatura, qualidade do ar, etc).

Geralmente, dados discretos são representados por vector que consiste de uma descrição da geometria ou forma da localidade e normalmente incluí variáveis com informações adicionais dessa localização (atributos). Campos espaciais (dados contínuos) são geralmente representados por raster.

2.1.1 - Dados vector

Os principais tipos de dados vectoriais são pontos, linhas e polígonos. Em todos os casos a geometria dessas estruturas de dados consiste em um conjunto de coordenadas (x,y).

Pontos: cada ponto tem um par de coordenadas e n variáveis associadas. Por exemplo, um ponto pode representar um ponto de amostragem e seus atributos podem incluir, nome de quem amostrou, data da amostragem, resultados geoquímicos, etc. É também possível combinar vários pontos numa estrutura de dados multiponto , com atributos em comum.

Linhas: Se referem a um conjunto de uma ou mais polilinhas (série de segmentos conectados). Exemplo, um rio e seus tributários. Linhas são representadas como um conjunto ordenado de coordenadas (nodes). O segmento de linha pode ser calculado ou plotado num mapa conectando-se os pontos.

Polígonos: Se referem a um conjunto de polilinhas fechadas. A geometria é bastante similar à das linhas mas para fechar o polígono o último ponto deve coincidir com o primeiro ponto. Polígonos podem apresentar buracos e não pode cruzar as linhas de suas bordas. Multipolígonos podem ser considerados como um simples polígono, ex. um arquipélago.

2.1.2 - Dados raster

Dados raster são comumente usados para representar valores contínuos. Ele divide uma região em uma malha (grid) de retângulos iguais (células ou pixels) que possuem um valor (ou ausência de valor) para as varáveis de interesse. O valor de uma célula raster pode ser representado pelo valor da média (ou maioria) da área que ela ocupa. No entanto, em alguns casos, o valor representa uma estimativa para o centro da célula (como se fossem multipontos distribuídos regularmente).

Diferentemente de vetor, a geometria dos dados raster não é explicitamente guardada na forma de coordenadas. Ela é implicitamente definida em sabendo-se a extensão espacial e o número de linhas e colunas nas quais a região é dividida. Essa informação nos dá a resolução espacial do raster (dimensões de cada célula).

Embora podemos pensar que um raster pode ser representado por uma série regularmente distribuída de pontos ou retângulos, seria muito ineficiente guardarmos as informações de coordenadas de cada ponto envolvido e aumentaria absurdamente o tempo de computação destes.

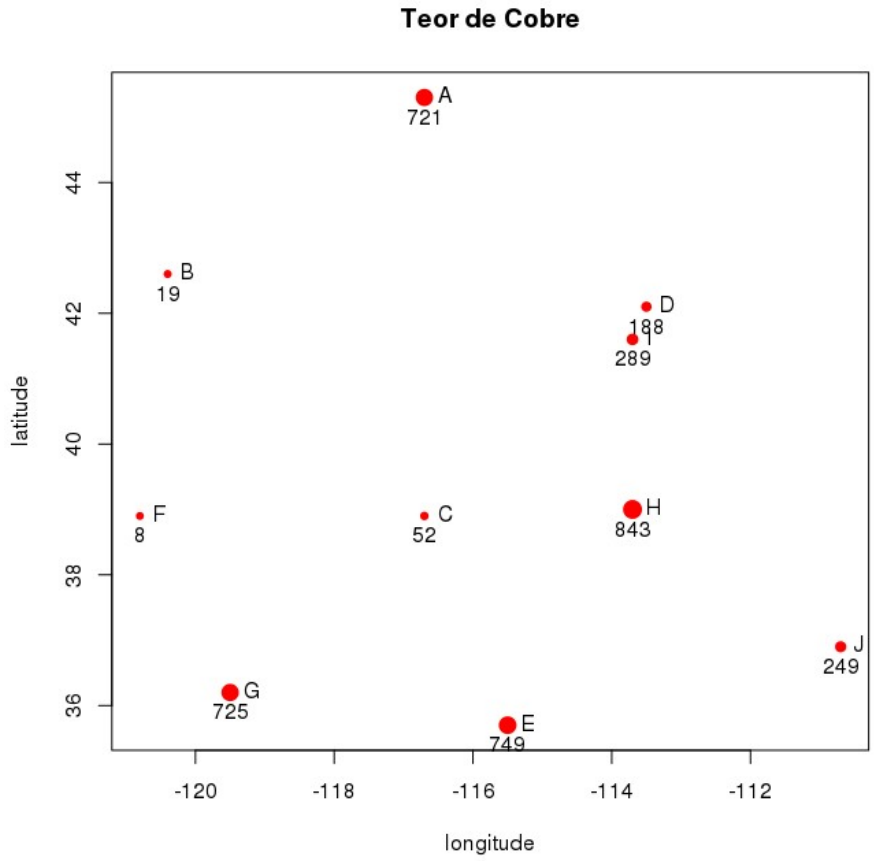
2.1.3 - Representação simples de dados espaciais

Os tipos básicos de dados em R são números, caracteres, lógico (TRUE ou FALSE) e fatores. Valores de um tipo único de dado pode ser combinado em arrays (incluindo vector e matriz). E variáveis de múltiplo tipos podem ser combinadas em um data.frame. Nós podemos representar dados espaciais usando estes tipos mas estes serão muito básicos. Vamos apresentar aqui, como exemplo, 10 pontos de amostragem nomeados de A a J com valores de teor de cobre.

```
> nome <- LETTERS[1:10]                    # gera vector de A a J
> longitude <- c(-116.7, -120.4, -116.7, -113.5, -115.5,
+                -120.8, -119.5, -113.7, -113.7, -110.7)
> latitude <- c(45.3, 42.6, 38.9, 42.1, 35.7, 38.9,
+               36.2, 39, 41.6, 36.9)
> pontos<-cbind(longitude, latitude)         #une long-lat em uma matrix
> #aleatoriamente adicionando valores de teor de cobre
> set.seed(0)
> teorCu<-(runif(10)*10)^3
```

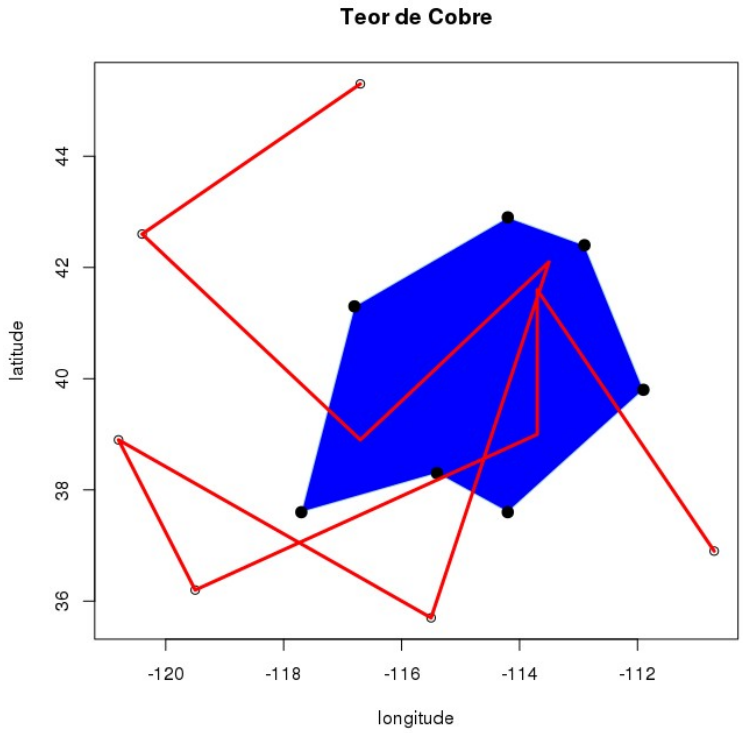
Um mapa de localização de pontos não é muito diferente de um gráfico de pontos x-y. Vamos plotar um mapa que mostra a localização dos pontos de amostragem e o tamanho dos pontos representará o teor de cobre. O tamanho é definido pelo argumento cex.

```
> plot(pontos, cex=1+teorCu/500, pch=20, col='red', main='Teor de Cobre')
> text(pontos,nome,pos=4)             # rótulo dos pontos na posição 4 (direita)
> text(pontos,as.character(round(teorCu)),pos=1) #rot. teores na pos 1 (abaixo)
```



Nós podemos adicionar vários conjuntos de pontos no gráfico (‘mapa’), incluindo linhas e polígonos.

```
> lon <- c(-116.8, -114.2, -112.9, -111.9, -114.2, -115.4, -117.7)
> lat <- c(41.3, 42.9, 42.4, 39.8, 37.6, 38.3, 37.6)
> x <- cbind(lon, lat)
> plot(pontos, main='Teor de Cobre')
> polygon(x, col='blue', border='light blue')
> points(x, cex=2, pch=20)
> lines(pontos, lwd=3, col='red')
```



Podemos de maneira simples criar um data.frame dos nossos pontos de amostragem para representar espacialmente nossos dados:

```
> cobre <- data.frame(longitude, latitude, nome, teorCu)
> cobre
   longitude latitude nome     teorCu
1      -116.7     45.3    A 721.003613
2      -120.4     42.6    B  18.716993
3      -116.7     38.9    C  51.530302
4      -113.5     42.1    D 187.988119
5      -115.5     35.7    E 749.127376
6      -120.8     38.9    F   8.203534
7      -119.5     36.2    G 725.093932
8      -113.7     39.0    H 843.038944
9      -113.7     41.6    I 288.539816
10     -110.7     36.9    J 248.993575
```

Mas muita informação ainda é necessária (sistema de coordenada, o que cada coluna representa, etc) para armazenar de forma eficiente dados espacias. Veremos a seguir como usar pacotes específicos que criam estruturas para lidar com dados espaciais efetivamente (os pacotes sp e raster).

2.2 – Dados Vetoriais em R usando o pacote sp

O pacote **sp** é o pacote central que dá suporte à análise de dados espaciais em R. ele define um conjunto de classes para representar dados espaciais. Uma classe define um tipo particular de dado.

Ele introduz um número de classes com o nome inciando com Spatial. Para os dados vector os tipos básicos são: SpatialPoints, SpatialLines e SpatialPolygons. Essas classes somente representam geometrias. Para guardar atributos também usamos as classes SpatialPointsDataFrame, SpatialLinesDataFrame e SpatialPolygonsDataFrame.

É possível criar objetos Spatial* (* para se referir a todas as classes) do início usando R. Geralmente usamos arquivos GIS para carregar estes dados, mas vamos usar alguns exemplos de forma demostrativa aqui para você ganhar familiaridade com as classes do pacote sp.

Vamos criar agora alguns objetos Spatial* do início, algum deles usamos no capítulo anterior.

2.2.1 - sp - SpatialPoints

Criamos os pontos como vetores combinados:

```
> longitude <- c(-116.7, -120.4, -116.7, -113.5, -115.5,
+                -120.8, -119.5, -113.7, -113.7, -110.7)
> latitude <- c(45.3, 42.6, 38.9, 42.1, 35.7, 38.9,
+               36.2, 39, 41.6, 36.9)
> pontos<-cbind(longitude, latitude) #une long-lat em uma matrix
```

E agora vamos criar um objeto SpatialPoints:

```
> library(sp)
> pts<-SpatialPoints(pontos)
```

Vamos ver que tipo de objeto ele é agora:

```
> class(pts)
[1] "SpatialPoints"
attr(,"package")
[1] "sp"
```

E o que tem dentro dele:

```
> showDefault(pts)
An object of class "SpatialPoints"
Slot "coords":
      longitude latitude
 [1,]    -116.7     45.3
 [2,]    -120.4     42.6
 [3,]    -116.7     38.9
 [4,]    -113.5     42.1
 [5,]    -115.5     35.7
 [6,]    -120.8     38.9
 [7,]    -119.5     36.2
 [8,]    -113.7     39.0
 [9,]    -113.7     41.6
[10,]    -110.7     36.9

Slot "bbox":
             min    max
longitude -120.8 -110.7
latitude    35.7   45.3

Slot "proj4string":
CRS arguments: NA
```

Podemos ver que o objeto SpatialPoints pts que criamos possuí as coordenadas que fornecemos, uma bbox (bounding box ou extensão espacial) das coordenadas, e temos também uma "proj4string" que guarda a referência do sistema de coordenadas (CRS que veremos mais adiante). Nós não fornecemos o CRS e por isso ele aparece como NA. Vamos então recriar o objeto com um CRS:

```
> ref<-CRS('+proj=longlat +datum=WGS84')
> pts<-SpatialPoints(pontos,proj4string=ref)
```

Vamos carregar o pacote raster somente para mostrar melhor a informação de pts.

```
> library(raster)
> pts
class       : SpatialPoints
features    : 10
extent      : -120.8, -110.7, 35.7, 45.3  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
```

Vamos agora carregar atributos no objeto criando um objeto SpatialPointsDataFrame a partir do objeto pts (SpatialPoints).

Primeiro criamos um data.frame com mesmo número de pontos do objeto pts:

```
> df <- data.frame(ID=1:nrow(pontos), teorCu=(latitude-30)^3)
> df
   ID   teorCu
1   1 3581.577
2   2 2000.376
3   3  704.969
4   4 1771.561
5   5  185.193
6   6  704.969
7   7  238.328
8   8  729.000
9   9 1560.896
10 10  328.509
```

E agora juntamos o data.frame com o nosso objeto pts:

```
> ptsdf <- SpatialPointsDataFrame(pts, data=df)
> ptsdf
class       : SpatialPointsDataFrame
features    : 10
extent      : -120.8, -110.7, 35.7, 45.3  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
variables   : 2
names       : ID,   teorCu
min values  :  1,  185.193
max values  : 10, 3581.577
```

Vamos ver o que temos dentro dele agora:

```
> showDefault(ptsdf)
An object of class "SpatialPointsDataFrame"
Slot "data":
   ID   teorCu
1   1 3581.577
2   2 2000.376
3   3  704.969
4   4 1771.561
5   5  185.193
6   6  704.969
7   7  238.328
8   8  729.000
9   9 1560.896
10 10  328.509

Slot "coords.nrs":
numeric(0)

Slot "coords":
      longitude latitude
 [1,]    -116.7     45.3
 [2,]    -120.4     42.6
 [3,]    -116.7     38.9
 [4,]    -113.5     42.1
 [5,]    -115.5     35.7
 [6,]    -120.8     38.9
 [7,]    -119.5     36.2
 [8,]    -113.7     39.0
 [9,]    -113.7     41.6
[10,]    -110.7     36.9

Slot "bbox":
            min    max
longitude -120.8 -110.7
latitude    35.7   45.3

Slot "proj4string":
CRS arguments:
 +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
```

2.2.2 - sp - SpatialLines e SpatialPolygons

Criar SpatialPoints foi fácil, Criar objetos SpatialLines e SpatialPolygons é um pouco mais difícil, mas ainda relativamente fácil usando as funções spLines e spPolygons (do pacote raster).

```
> lon <- c(-116.8, -114.2, -112.9, -111.9, -114.2, -115.4, -117.7)
> lat <- c(41.3, 42.9, 42.4, 39.8, 37.6, 38.3, 37.6)
> lonlat <- cbind(lon, lat)
> lns<-spLines(lonlat,crs=ref)
> lns
class       : SpatialLines
features    : 1
extent      : -117.7, -111.9, 37.6, 42.9  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
```

```
> pols<-spPolygons(lonlat,crs=ref)
> pols
```

```
class       : SpatialPolygons
features    : 1
extent      : -117.7, -111.9, 37.6, 42.9  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
```

A estrutura da classe SpatialPolygons e de certa maneira bem complicado pois precisa acomodar a possibilidade de múltiplos polígonos, cada um dos quais com subpolígonos e podendo haver buracos ou vazios neles.

```
> str(pols)
Formal class 'SpatialPolygons' [package "sp"] with 4 slots
  ..@ polygons   :List of 1
  .. ..$ :Formal class 'Polygons' [package "sp"] with 5 slots
  .. .. .. ..@ Polygons :List of 1
  .. .. .. .. ..$ :Formal class 'Polygon' [package "sp"] with 5 slots
  .. .. .. .. .. .. ..@ labpt  : num [1:2] -114.7 40.1
  .. .. .. .. .. .. ..@ area   : num 19.7
  .. .. .. .. .. .. ..@ hole   : logi FALSE
  .. .. .. .. .. .. ..@ ringDir: int 1
  .. .. .. .. .. .. ..@ coords : num [1:8, 1:2] -117 -114 -113 -112 -114 ...
  .. .. .. ..@ plotOrder: int 1
  .. .. .. ..@ labpt    : num [1:2] -114.7 40.1
  .. .. .. ..@ ID       : chr "1"
  .. .. .. ..@ area     : num 19.7
  ..@ plotOrder  : int 1
  ..@ bbox       : num [1:2, 1:2] -117.7 37.6 -111.9 42.9
  .. ..- attr(*, "dimnames")=List of 2
  .. .. ..$ : chr [1:2] "x" "y"
  .. .. ..$ : chr [1:2] "min" "max"
  ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
  .. .. ..@ projargs: chr "+proj=longlat +datum=WGS84 +ellps=WGS84
+towgs84=0,0,0"
```

Felizmente não precisamos nos preocupar com isso pois existem várias funções para interpretar isso. O ponto importante aqui foi mostrar que os objetos criados possuem geometrias (coordenadas), nome do sistema de coordenada e atributos.

2.3 – Dados Raster em R

O pacote **sp** dá suporte para dados raster (em grid) com as classes SpatialGridDataFrame e SpatialPixelsDataFrame. Entretanto nós vamos focar nas classes do pacote **raster** para os dados raster. O pacote **raster** é construído em volta de várias classes mas as classes RasterLayer, RasterBrick e RasterStack são as mais importantes. Quando falamos de métodos que podem operar em todas as três classes vamos usar o termo objetos Raster*.

O pacote raster possui funções para criar, ler, manipular e escrever dados raster. Ele fornece, entre outras coisas, funções de manipulação de dados raster que podem facilmente ser usados para criar funções mais específicas.

2.3.1 - RasterLayer

Um objeto RasterLayer representa um dado raster de uma só camada ou banda (variável). Ele sempre armazena um número fundamental de parâmetros que o descrevem. Estes parâmetros incluem o número de linhas e colunas, a extensão espacial, o sistema de coordenada (CRS). Além disso, um objeto RasterLayer pode guardar informação sobre o arquivo no qual os valores das células estão armazenados (caso exista). Um objeto RasterLayer pode também armazenar os valores das células na memória.

Aqui vamos criar um objeto RasterLayer do início. Mas quase sempre eles são criados a partir de um arquivo.

```
> library(raster)
Carregando pacotes exigidos: sp
> rl<-raster(ncol=100,nrow=100,xmx=301000,xmn=300000,ymn=7998000,ymx=7999000)
> rl
class       : RasterLayer
dimensions  : 100, 100, 10000  (nrow, ncol, ncell)
resolution  : 10, 10  (x, y)
extent      : 3e+05, 301000, 7998000, 7999000  (xmin, xmax, ymin, ymax)
coord. ref. : NA
```

Vamos agora atribuir um CRS ao objeto:

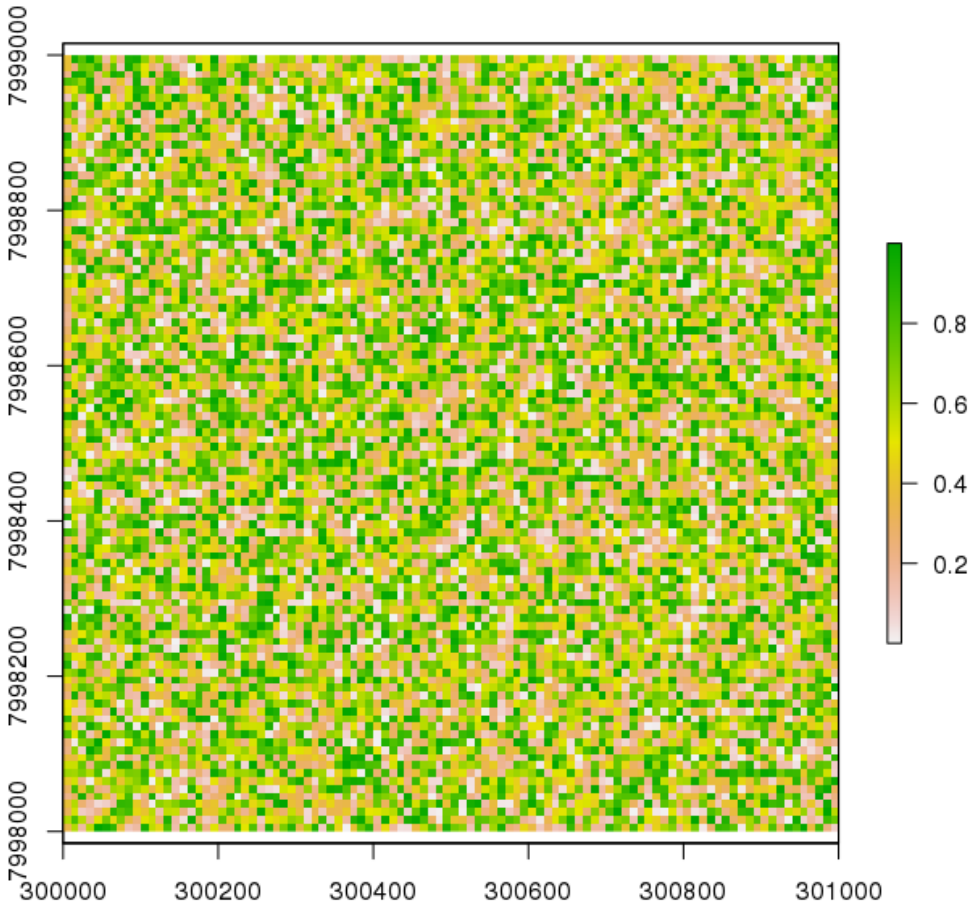
```
> crs(rl)<-CRS('+init=epsg:32723')
> rl
class       : RasterLayer
dimensions  : 100, 100, 10000  (nrow, ncol, ncell)
resolution  : 10, 10  (x, y)
extent      : 3e+05, 301000, 7998000, 7999000  (xmin, xmax, ymin, ymax)
coord. ref. : +init=epsg:32723 +proj=utm +zone=23 +south +datum=WGS84 +units=m
+no_defs +ellps=WGS84 +towgs84=0,0,0
```

Vamos agora inserir valores no nosso objeto aleatoriamente:

```
> values(rl)<-runif(ncell(rl))
> rl
class       : RasterLayer
dimensions  : 100, 100, 10000  (nrow, ncol, ncell)
resolution  : 10, 10  (x, y)
extent      : 3e+05, 301000, 7998000, 7999000  (xmin, xmax, ymin, ymax)
coord. ref. : +init=epsg:32723 +proj=utm +zone=23 +south +datum=WGS84 +units=m
+no_defs +ellps=WGS84 +towgs84=0,0,0
data source : in memory
names       : layer
values      : 3.881566e-05, 0.9998762  (min, max)
```

E plotar o nosso objeto raster

```
> plot(rl)
```



2.3.2 - RasterStack e RasterBrick

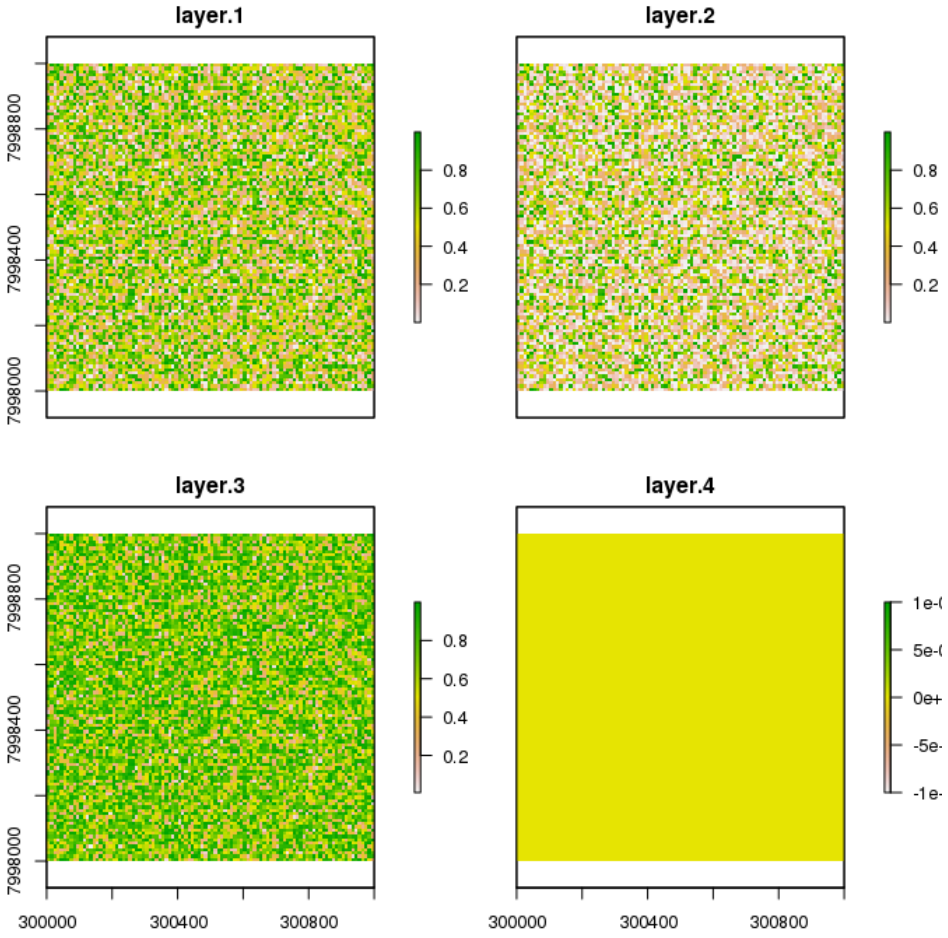
É bastante comum analisarmos objetos raster de uma só camada. Entretanto, em muitos casos, objetos raster de multivariáveis (mais de uma banda) são usados. O pacote raster tem duas classes para dados multicamadas, o RasterStack e o RasterBrick. A principal diferença entre as duas classes é que a RasterBrick só pode ser ligada a um único arquivo multicamada. Por outro lado, RasterStack pode ser formado a partir de arquivos separados e/ou por algumas bandas de um arquivo único.

Assim, a principal diferença é que um objeto RasterStack é uma coleção solta de objetos RasterLayer proveniente de várias fontes (mas com mesma extensão e resolução) enquanto um objeto RasterBrick somente pode apontar para uma única fonte.

Aqui temos um exemplo de como podemos criar um RasterStack a partir de várias camadas:

```
> rl2<-rl*rl
> rl3<-sqrt(rl)
> rl4<-rl*0
> rs<-stack(rl,rl2,rl3,rl4)
> rs
class       : RasterStack
dimensions  : 100, 100, 10000, 4  (nrow, ncol, ncell, nlayers)
resolution  : 10, 10  (x, y)
```

```
extent      : 3e+05, 301000, 7998000, 7999000  (xmin, xmax, ymin, ymax)
coord. ref. : +init=epsg:32723 +proj=utm +zone=23 +south +datum=WGS84 +units=m
+no_defs +ellps=WGS84 +towgs84=0,0,0
names       :      layer.1,      layer.2,      layer.3,      layer.4
min values  : 3.881566e-05, 1.506656e-09, 6.230222e-03, 0.000000e+00
max values  :    0.9998762,    0.9997524,    0.9999381,    0.0000000
> plot(rs)
```



E podemos criar um RasterBrick a partir deste único RasterStack criado acima:

```
> rb<-brick(rs)
> rb
class       : RasterBrick
dimensions  : 100, 100, 10000, 4  (nrow, ncol, ncell, nlayers)
resolution  : 10, 10  (x, y)
extent      : 3e+05, 301000, 7998000, 7999000  (xmin, xmax, ymin, ymax)
coord. ref. : +init=epsg:32723 +proj=utm +zone=23 +south +datum=WGS84 +units=m
+no_defs +ellps=WGS84 +towgs84=0,0,0
data source : in memory
names       :      layer.1,      layer.2,      layer.3,      layer.4
min values  : 3.881566e-05, 1.506656e-09, 6.230222e-03, 0.000000e+00
max values  :    0.9998762,    0.9997524,    0.9999381,    0.0000000
```

2.4 – Lendo e escrevendo dados espaciais

Ler e escrever arquivos é muito simples usando R, vamos ver aqui como ler e escrever arquivos vector de vários formatos e arquivos raster usando o pacote rgdal.

2.4.1 - Lendo arquivos vector - sp

O pacote rgdal tem a função readOGR para ler arquivos de vários formatos, podemos ler o arquivo do diretório de trabalho usando:

```
> library(rgdal)
> sh2<-readOGR(dsn=".",layer="minasgerais_lito")
OGR data source with driver: ESRI Shapefile
Source: "/home/andre/livroR", layer: "minasgerais_lito"
with 5676 features
It has 26 fields
> library(raster)
> sh2
class       : SpatialPolygonsDataFrame
features    : 5676
extent      : -51.03835, -39.8611, -22.91533, -14.22754  (xmin, xmax, ymin,
ymax)
crs         : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
variables   : 26
names       : SIGLA_UNID, NOME_UNIDA, HIERARQUIA,
LITOTIPO1,
LITOTIPO2,       CLASSE_ROC, COD_DOM,
DOMINIO, COD_UNIGEO,
UNIGEO,                      DEF_TEC,             CIS_FRAT,   ASPECTO,
INTEMP_F,      INTEMP_Q, ...
min values  :       A2gm,    Acaiaca,   Complexo,
Aglomerado, Laterita, Aglomerado, Formação Ferrífera Bandada, Metabasalto,
Metabasalto Komatiítico, Metatufo, Metaconglomerado, Metachert,
Ígena/Sedimentar,      DC,
Domínio das coberturas Cenozóicas Detrito-Lateríticas,        DCa, Alternância
irregular entre camadas de sedimentos de composição diversa (Arenito, siltito,
argilito e cascalho).,                      Ausente,              Ausente,
Acamadada,         Baixa,         Baixa, ...
max values  :        Qdi,      Viana,    Unidade, Xisto, Quartzo-mica xisto,
Grafita xisto,
Xisto, Quartzito, Metassiltito,       Sedimentar,     DVM, Domínio dos
sedimentos indiferenciados Cenozóicos relacionados a Retrabalhamento de outras
rochas, geralmente associados a superfícies de aplainamento,       DVMb,
Vulcânicas básica., Pouco a moderadamente dobrada, Zonas de cisalhamento,
Xistosa, Não se aplica, Não se aplica, ...
```

2.4.2 - Escrevendo arquivos vector - sp

Usamos a função writeOGR conforme abaixo podemos gravar o objeto Spatial* no arquivo com o nome dado por layer e usando o driver especificado por driver.

```
> writeOGR(sh2, dsn=".", layer="teste", driver="ESRI Shapefile")
```

2.4.3 - Lendo arquivos raster

Usando o pacote raster podemos abrir diretamente arquivos raster usando as funções raster, stack ou brick.

A função brick abre imagens que contenham mais de uma banda,

```
> library(raster)
> b<-brick('fcc12_11_8.tif')
> b
class      : RasterBrick
dimensions : 5490, 5490, 30140100, 3  (nrow, ncol, ncell, nlayers)
resolution : 20, 20  (x, y)
extent     : 499980, 609780, 7990240, 8100040  (xmin, xmax, ymin, ymax)
crs        : +proj=utm +zone=23 +south +ellps=WGS84 +towgs84=0,0,0,0,0,0,0
+units=m +no_defs
source     : /home/andre/livroR/fcc12_11_8.tif
names      : fcc12_11_8.1, fcc12_11_8.2, fcc12_11_8.3
min values :            0,            0,            0
max values :        24993,        12590,         8040
```

A função raster abre imagens com uma única banda,

```
> r<-raster('b2lr.tif')
> r
class      : RasterLayer
dimensions : 5490, 5490, 30140100  (nrow, ncol, ncell)
resolution : 20, 20  (x, y)
extent     : 499980, 609780, 7990240, 8100040  (xmin, xmax, ymin, ymax)
crs        : +proj=utm +zone=23 +south +datum=WGS84 +units=m +no_defs
+ellps=WGS84 +towgs84=0,0,0
source     : /home/andre/livroR/b2lr.tif
names      : b2lr
values     : 0, 65535  (min, max)
```

A função stack abre imagens de fontes diferentes mas com mesma resolução e área.

```
> s<-stack('b2lr.tif','tcc4_3_2.tif','fcc12_11_8.tif')
> s
class      : RasterStack
dimensions : 5490, 5490, 30140100, 7  (nrow, ncol, ncell, nlayers)
resolution : 20, 20  (x, y)
extent     : 499980, 609780, 7990240, 8100040  (xmin, xmax, ymin, ymax)
crs        : +proj=utm +zone=23 +south +datum=WGS84 +units=m +no_defs
+ellps=WGS84 +towgs84=0,0,0
names      :  b2lr, tcc4_3_2.1, tcc4_3_2.2, tcc4_3_2.3, fcc12_11_8.1,
fcc12_11_8.2, fcc12_11_8.3
min values :     0,          0,          0,          0,            0,
0,            0
max values : 65535,       8386,       8162,       7938,        24993,
12590,         8040
```

2.4.4 - Escrevendo um arquivo raster

Use a função writeRaster para gravar um arquivo raster a partir de um objeto do tipo RasterLayer, RasterStack ou RasterBrick.

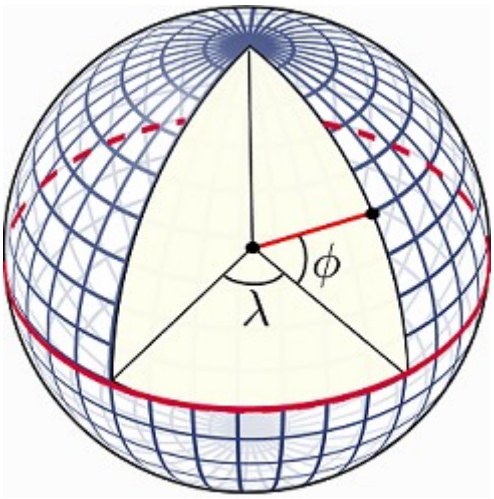
```
> writeRaster(s, 'output.tif',format='GTiff', overwrite=TRUE)
```

2.5 – Sistema de Referência de Coordenadas (CRS)

Um aspecto muito importante para a análise de dados espaciais é o sistema de referências de coordenadas (CRS) que é usado. Uma coordenada sem o devido CRS não faz muito sentido e vamos aqui falar um pouco sobre isso.

2.5.1 - Sistema de referência de coordenadas

A terra tem um formato irregular que se aproxima a uma esfera. O sistema natural de referência de coordenadas para dados geográficos é longitude/latitude. Este sistema é angular onde Φ é a latitude e λ é a longitude:



Obviamente não podemos medir estes ângulos diretamente mas podemos estimar eles usando um modelo da forma da terra. Este modelo é chamado ‘datum’. O datum mais simples são os esferoides (esferas achatadas nos polos e mais largas no equador). O datum mais comum usado é o WGS-84 (World Geodesic System 1984). Outros data locais existem para melhor registrar localidades para uma região ou país específico.

2.5.2 - Projeções

O maior problema em análise espacial e cartografia é como transformar este sistema angular tridimensional para um sistema planar bidimensional (cartesiano). Um sistema planar é mais fácil para efetuar operações e essencial para se criar mapas. Os diferentes tipos de sistema de referência de coordenadas são referidos como projeções tais como Mercator, Albers, Lambert, etc.

Não existe uma projeção melhor, algumas são melhores para criar mapas do mundo inteiro, outras melhores para criar mapas de áreas pequenas somente. Uma das características mais importantes de uma projeção é se ela é equalárea (escala constante) ou conformal (da forma que enxergamos). Nenhum mapa bidimensional pode ser as duas ao mesmo tempo e algumas não são nem equalárea e nem conformais.

2.5.3 - Notação

Uma CRS planar é definida por uma projeção, um datum e um conjunto de parâmetros (número de parâmetros depende da projeção). As vezes não é fácil documentar uma projeção usada. R usa PROJ.4.

Existe também o sistema de códigos EPSG que usam um simples número para atribuir uma projeção. Este sistema é bem prático e é comumente usado em banco de dados.

Particularmente prefiro usar EPSG em vez de PROJ.4 e é muito mais prático trabalhar somente com um datum para evitar erros. Sugiro usar sempre WGS-84 para longitude/latitude bem como para UTM.

Os códigos EPSG para WGS-84 são:
Longitude/Latitude : **4326**
UTM Norte: **32601** (Zona 1) até **32660** (zona 60)
UTM Sul: **32701** (Zona1) até **32760** (zona 60)

2.5.4 - Designando uma CRS a um objeto

A dois capítulo atrás fizemos isso quando criamos um objeto raster. Usamos este método para simplesmente dizer qual CRS o objeto é. **LEMBRE QUE ISSO NÃO VAI REPROJETAR O OBJETO PARA O CRS USADO.**

```
> library(raster)
Carregando pacotes exigidos: sp
> rl<-raster(ncol=100,nrow=100,xmx=301000,xmn=300000,ymn=7998000,ymx=7999000)
> crs(rl)<-CRS('+init=epsg:32723')
```

2.5.5 - Reprojetando dados vector - sp

Reprojetando objeto vector de WGS-84 longitude latitude para WGS-84 UTM Zona 23 usando função spTransform:

```
> library(raster)
> library(rgdal)
> lay<-readOGR(dsn='.',layer='teste')
OGR data source with driver: ESRI Shapefile
Source: "/home/andre/livroR", layer: "teste"
with 5676 features
It has 26 fields
> lay
class       : SpatialPolygonsDataFrame
features    : 5676
extent      : -51.03835, -39.8611, -22.91533, -14.22754  (xmin, xmax, ymin,
ymax)
crs         : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
variables   : 26
names       : SIGLA_UNID, NOME_UNIDA, HIERARQUIA,
LITOTIPO1,
LITOTIPO2,       CLASSE_ROC, COD_DOM,
DOMINIO, COD_UNIGEO,
UNIGEO,                      DEF_TEC,             CIS_FRAT,   ASPECTO,
INTEMP_F,      INTEMP_Q, ...
min values  :       A2gm,    Acaiaca,   Complexo,
Aglomerado, Laterita, Aglomerado, Formação Ferrífera Bandada, Metabasalto,
Metabasalto Komatiítico, Metatufo, Metaconglomerado, Metachert,
Ígena/Sedimentar,      DC,
Domínio das coberturas Cenozóicas Detrito-Lateríticas,        DCa, Alternância
irregular entre camadas de sedimentos de composição diversa (Arenito, siltito,
argilito e cascalho).,                      Ausente,              Ausente,
Acamadada,         Baixa,         Baixa, ...
max values  :        Qdi,      Viana,    Unidade, Xisto, Quartzo-mica xisto,
Grafita xisto,
Xisto, Quartzito, Metassiltito,       Sedimentar,     DVM, Domínio dos
```

```
sedimentos indiferenciados Cenozóicos relacionados a Retrabalhamento de outras
rochas, geralmente associados a superfícies de aplainamento,       DVMb,
Vulcânicas básica., Pouco a moderadamente dobrada, Zonas de cisalhamento,
Xistosa, Não se aplica, Não se aplica, ...
> lay2<-spTransform(lay,"+init=epsg:32723")
> lay2
class       : SpatialPolygonsDataFrame
features    : 5676
extent      : -133630.7, 1050148, 7465399, 8426966  (xmin, xmax, ymin, ymax)
crs         : +init=epsg:32723 +proj=utm +zone=23 +south +datum=WGS84 +units=m
+no_defs +ellps=WGS84 +towgs84=0,0,0
variables   : 26
names       : SIGLA_UNID, NOME_UNIDA, HIERARQUIA,
LITOTIPO1,
LITOTIPO2,       CLASSE_ROC, COD_DOM,
DOMINIO, COD_UNIGEO,
UNIGEO,                      DEF_TEC,             CIS_FRAT,   ASPECTO,
INTEMP_F,      INTEMP_Q, ...
min values  :       A2gm,    Acaiaca,   Complexo,
Aglomerado, Laterita, Aglomerado, Formação Ferrífera Bandada, Metabasalto,
Metabasalto Komatiítico, Metatufo, Metaconglomerado, Metachert,
Ígena/Sedimentar,      DC,
Domínio das coberturas Cenozóicas Detrito-Lateríticas,        DCa, Alternância
irregular entre camadas de sedimentos de composição diversa (Arenito, siltito,
argilito e cascalho).,                       Ausente,              Ausente,
Acamadada,         Baixa,         Baixa, ...
max values  :        Qdi,      Viana,    Unidade, Xisto, Quartzo-mica xisto,
Grafita xisto,
Xisto, Quartzito, Metassiltito,       Sedimentar,     DVM, Domínio dos
sedimentos indiferenciados Cenozóicos relacionados a Retrabalhamento de outras
rochas, geralmente associados a superfícies de aplainamento,       DVMb,
Vulcânicas básica., Pouco a moderadamente dobrada, Zonas de cisalhamento,
Xistosa, Não se aplica, Não se aplica, ...
```

2.5.6 - Reprojetando dados raster

Usamos a função projectRaster para converter objeto raster de WGS-84 UTM zona 23 para Longitude/latitude WGS-84:

```
> library(raster)
Carregando pacotes exigidos: sp
> s<-raster('demcut.tif')
> s
class      : RasterLayer
dimensions : 299, 299, 89401  (nrow, ncol, ncell)
resolution : 30.10033, 30.10033  (x, y)
extent     : 5e+05, 509000, 7990240, 7999240  (xmin, xmax, ymin, ymax)
crs        : +proj=utm +zone=23 +south +datum=WGS84 +units=m +no_defs
+ellps=WGS84 +towgs84=0,0,0
source     : /home/andre/livroR/demcut.tif
names      : demcut
values     : -32768, 32767  (min, max)
> s2<-projectRaster(s,crs='+init=epsg:4326')
> s2
class      : RasterLayer
dimensions : 309, 309, 95481  (nrow, ncol, ncell)
resolution : 0.000285, 0.000272  (x, y)
extent     : -45.00142, -44.91336, -18.17825, -18.0942  (xmin, xmax, ymin,
ymax)
crs        : +init=epsg:4326 +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84
+towgs84=0,0,0
```

```
source     : memory
names      : demcut
values     : 687.1117, 846.3503  (min, max)
data source : in memory
names       : demcut
values      : 45.77902, 1369.74  (min, max)
```

Dica: reprojetar raster em ambiente R é muito demorado, dependendo do tamanho do raster, use gdalwarp para reprojetar.

2.6 – Manipulação de dados vector – pacote sp

Agora vamos aprender como trabalhar com dados vetoriais para extrair e modificar informações destes usando o pacote sp. Vamos aqui mostrar como isso é feito usando sempre um objeto SpatialPoints* e um objeto SpatialPolygons* e posteriormente usamos dois objetos SpatialPolygon* para efetuar operações entre objetos.

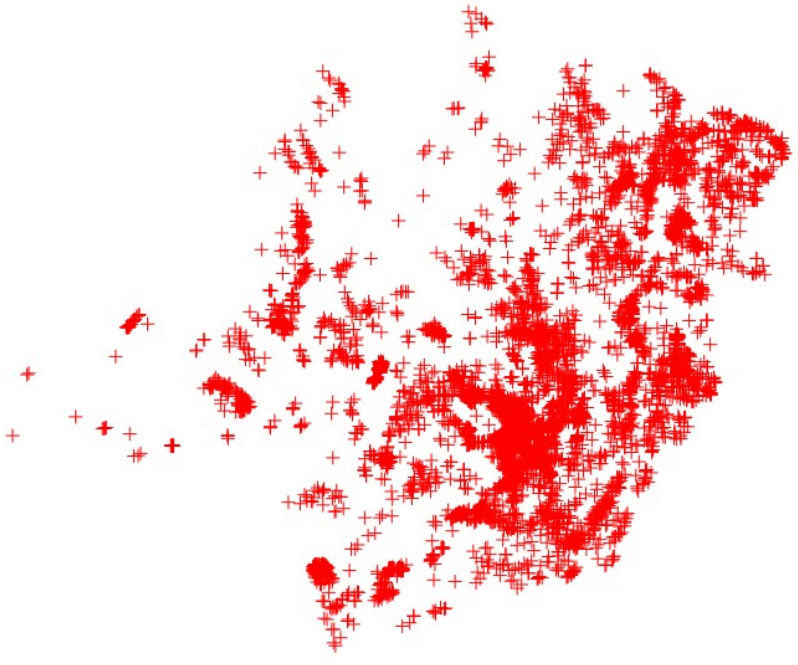
2.6.1 - Carregando arquivo , enxugando dados e gravando novo arquivo

Dentro do nosso diretório de trabalho carregamos dois arquivos, um de pontos e outro de polígonos e abrimos estes usando a função readOGR do pacote rgdal.

```
> library(rgdal)
> library(raster) # para melhor mostrar informações dos objetos
> pnt<-readOGR(dsn='.',layer='minasgerais_recmin')
OGR data source with driver: ESRI Shapefile
Source: "/home/andre/livroR", layer: "minasgerais_recmin"
with 6818 features
It has 24 fields
> plgn<-readOGR(dsn='.',layer='se23_lito')
OGR data source with driver: ESRI Shapefile
Source: "/home/andre/livroR", layer: "minasgerais_lito"
with 5676 features
It has 26 fields
```

Vamos agora plotar esses dois objetos:

```
> par(mai=c(0,0,0,0))
> plot(pnt,col='red')
> par(mai=c(0,0,0,0))
> plot(plgn)
```



*Objeto SpatialPoints**

*Objeto SpatialPolygons**

Extraímos a geometria e atributos desses objetos da seguinte forma, criando um data.frame para armazenar os atributos e criando uma matriz para armazenar a geometria (pouco usado).

```
> #objeto SpatialPoints*
> pnt.df<-data.frame(pnt)
> head(pnt.df,n=1L)
  data_cadas subst_prin abreviacao           associacao associac_1 extrmin_x
1 2003/11/26       OURO         Au Ouro, Pirita, Quartzo         Au Filoneana
  grau_de_in                      metodo_geo  textura_mi        origem
1       <NA> Levantamento em carta 1:100.000 Disseminada São Francisco
            toponimia status_eco   municipio uf
1 Ribeirão da Passagem       <NA> Ouro Branco MG
                                    tipologia    classe_uti subst_secu
1 Depósitos relacionados a processos metamórficos Metais nobres       <NA>
       rocha_hosp rocha_enca             classe_gen
1 Veio de quartzo      Xisto Metamórfica hidrotermal
                    modelo_dep tipos_alte             grupo
1 Concentração metálica em veios       <NA> MINERAIS METÁLICOS
          classe_u_1 coords.x1 coords.x2 optional
1 Minerais Metálicos -43.75472 -20.56444     TRUE

> pnt.geom<-geom(pnt)
> head(pnt.geom,n=1L)
     object         x         y
[1,]      1 -43.75472 -20.56444

> # objeto SpatialPolygons*
> plgn.df<-data.frame(plgn)
> head(plgn.df,n=1L)
  SIGLA_UNID        NOME_UNIDA HIERARQUIA  LITOTIPO1
0      NP2rf Ribeirão da Folha   Formação Mica xisto
```

```
LITOTIPO2
0 Rocha Calcissilicática, Formação Ferrífera Bandada, Metarcóseo, Metamáfica,
Metaultramafito
   CLASSE_ROC COD_DOM
0 Metamórfica   DSVP2

DOMINIO
0 Domínio das Sequências Vulcanossedimentares Proterozóicas dobradas
metamorfizadas de baixo a alto grau
  COD_UNIGEO
0   DSVP2vfc

UNIGEO
0 Metacherts, metavulcânicas, formações ferríferas e/ou formações
maganesíferas, metacalcários, metasedimentos arenosos e silticos argilosos.
                        DEF_TEC                          CIS_FRAT
0 Pouco a moderadamente dobrada Pouco a moderadamente fraturada
                         ASPECTO                                  INTEMP_F
0 Anisotrópica Acamadada/Xistosa Baixa a alta na horizontal e na vertical
                                INTEMP_Q              PORO_PRI
0 Baixa a alta na horizontal e na vertical Variável - (0 a >30%)
                               TEXTURA                          GR_COER
0 Variável de arenoso a argilo-siltoso Variável na horizontal e vertical
         LITO_HIDRO COD_REL             RELEVO DECLIVIDAD          AMPL_TOPO
0 Granular/fissural     R4c Domínio Montanhoso   25 a 45° 300 a 2.000 metros
       GEO_REL LEGENDA  OBS
0 DSVP2vfc/R4c      33 <NA>
>
> plgn.geom<-geom(plgn)
> head(plgn.geom,n=1L)
     object part cump hole         x         y
[1,]      1    1    1    0 -41.83287 -16.17763
```

Enxugando dados removendo algumas variáveis:

```
> plgn<-plgn[,-c(5,8:26)]
> head(plgn,n=1L)
  SIGLA_UNID        NOME_UNIDA HIERARQUIA  LITOTIPO1  CLASSE_ROC COD_DOM
0      NP2rf Ribeirão da Folha   Formação Mica xisto Metamórfica   DSVP2

> pnt<-pnt[,-c(1,5,6,8,9,10,15:24)]
> head(pnt,n=1L)
subst_prin abreviacao            associacao grau_de_in            toponimia
1       OURO         Au Ouro, Pirita, Quartzo       <NA> Ribeirão da Passagem
  status_eco   municipio uf
1       <NA> Ouro Branco MG
```

Agora gravamos os arquivos na forma enxuta:

```
>  writeOGR(pnt, dsn=".", layer="minasgerais_recminE", driver="ESRI Shapefile")
>  writeOGR(plgn, dsn=".", layer="minasgerais_litoE", driver="ESRI Shapefile")
```

Os arquivos foram gravados no diretório de trabalho com os nomes minasgerais_recminE.shp e minasgerais_litoE.shp.

2.6.2 - Carregando Arquivos, vendo registros, alterando/adicionando variável e unindo data.frame

Vamos agora carregar dois arquivos do tipo GMT para criar dois objetos Spatial*:

```
> library(rgdal)
> library(raster)
> picos<-readOGR(dsn='SE-23-naturalpic.gmt',layer='SE-23-naturalpic')
OGR data source with driver: OGR_GMT
Source: "/home/andre/livroR/SE-23-naturalpic.gmt", layer: "SE-23-naturalpic"
with 70 features
It has 4 fields
> head(picos,n=3L)
      osm_id code fclass              name
1  267115440 4111   peak
2  559013004 4111   peak    Pico do Itambé
3 1906457751 4111   peak Morro do Jerônimo

> parques<-readOGR(dsn='SE-23-landuse-anat.gmt',layer='SE-23-landuse-anat')
Source:"/home/andre/livroR/SE-23-landuse-anat.gmt", layer: "SE-23-landuse-anat"
with 26 features
It has 4 fields
> head(parques,n=3L)
     osm_id code         fclass                         name
0   5121285 7210 nature_reserve       Parque das Mangabeiras
1 161118673 7210 nature_reserve  Jardim Botânico de Brasília
2 183216538 7210 nature_reserve Parque Estadual do Sumidouro
```

Vamos listar uma variável:

```
> parques$name
 [1] Parque das Mangabeiras
 [2] Jardim Botânico de Brasília
 [3] Parque Estadual do Sumidouro
 [4] Parque Nacional das Sempre-Vivas
 [5]
 [6] Área de Proteção Especial Ribeirão do Urubu
 [7] Área de Proteção Especial Córrego Cercadinho
 [8] Área de Proteção Especial Bacia Vargem das Flores
 [9] Área de Proteção Especial UHE Florestal
[10] Área de Proteção Especial Bacia do Ribeirão Laje
[11] Área de Proteção Especial Córrego Feio e Fundo
[12] Área de Proteção Especial UHE de Peti
[13] Estação Ecológica de Pirapitinga
[14] Floresta Nacional de Paraopeba
[15] Terra Indígena Fazenda Guarani
[16] Parque Nacional da Serra do Gandarela
[17] Parque Estadual do Biribiri
[18] Monumento Natural Estadual Várzea do Lageado e Serra do Raio
[19] Parque Estadual do Pico do Itambé
[20] Parque Estadual Serra do Intendente
[21] Parque Estadual da Lapa Grande
[22] Parque Nacional da Serra do Cipó
[23] Fazendinha Regional Nordeste
[24]
[25]
[26]
23 Levels:  ... Terra Indígena Fazenda Guarani
```

Obtendo informações gerais de cada objeto:

```
> parques
class       : SpatialPolygonsDataFrame
features    : 26
extent      : -47.98418, -42.05772, -20, -16  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
variables   : 4
names       :    osm_id, code,         fclass,                          name
min values  : 161118673, 7210, nature_reserve,
max values  :   7659215, 7210, nature_reserve, Terra Indígena Fazenda Guarani

> picos
class       : SpatialPointsDataFrame
features    : 70
extent      : -47.78814, -42.58098, -19.99988, -16.05905  (xmin, xmax, ymin,
ymax)
coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
variables   : 4
names       :     osm_id, code, fclass,             name
min values  : 1906457751, 4111,   peak,
max values  : 5600311623, 4111,   peak, Serra dos Óculos
```

Ou de uma variável específica:

```
> parques['fclass']
class       : SpatialPolygonsDataFrame
features    : 26
extent      : -47.98418, -42.05772, -20, -16  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
variables   : 1
names       :         fclass
min values  : nature_reserve
max values  : nature_reserve

> picos['name']
class       : SpatialPointsDataFrame
features    : 70
extent      : -47.78814, -42.58098, -19.99988, -16.05905  (xmin, xmax, ymin,
ymax)
coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
variables   : 1
names       :             name
min values  :
max values  : Serra dos Óculos
```

Para adicionar uma nova variável (atributo) ao objeto usamos:

```
> parques$valorNovo<-round(runif(length(parques), 100, 1000))
> parques
class       : SpatialPolygonsDataFrame
features    : 26
extent      : -47.98418, -42.05772, -20, -16  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
variables   : 5
names       :    osm_id, code,         fclass,                          name,
valorNovo
min values  : 161118673, 7210, nature_reserve,                              ,
114
max values  :   7659215, 7210, nature_reserve, Terra Indígena Fazenda Guarani,
998

> picos$Novo<-round(runif(length(picos), 1, 10))
```

```
> picos
class       : SpatialPointsDataFrame
features    : 70
extent      : -47.78814, -42.58098, -19.99988, -16.05905  (xmin, xmax, ymin,
ymax)
coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
variables   : 5
names       :     osm_id, code, fclass,             name, Novo
min values  : 1906457751, 4111,   peak,                ,    1
max values  : 5600311623, 4111,   peak, Serra dos Óculos,   10
```

Podemos visualizar um registro usando:

```
> picos[which(picos$name == 'Serra dos Óculos'),]
class       : SpatialPointsDataFrame
features    : 1
extent      : -46.76856, -46.76856, -18.79662, -18.79662  (xmin, xmax, ymin,
ymax)
coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
variables   : 5
names       :     osm_id, code, fclass,             name, Novo
value       : 3882762430, 4111,   peak, Serra dos Óculos,    1
```

Ou usando o comando abaixo para ver somente os registros:

```
> data.frame(picos[which(picos$name=='Morro da Lagoa'),])
       osm_id code fclass           name Novo coords.x1 coords.x2 optional
62 4743119890 4111   peak Morro da Lagoa    3 -43.46991 -19.67411     TRUE
>
> data.frame(picos[which(picos$Novo>8),])
       osm_id code fclass             name Novo coords.x1 coords.x2 optional
4  1962488318 4111   peak Serra da Piedade   10 -43.67642 -19.82256     TRUE
11 2264978243 4111   peak Morro Joana Vais   10 -47.15820 -19.99161     TRUE
13 2337673107 4111   peak   Morro da Serra   10 -47.51455 -19.96493     TRUE
15 2344861948 4111   peak   Morro do Facão    9 -46.99496 -19.72115     TRUE
16 2344861949 4111   peak   Morro do Forno    9 -46.99474 -19.68411     TRUE
22 3052378045 4111   peak   Morro Queimado    9 -43.44692 -19.97229     TRUE
42 3926034336 4111   peak     Morro do Baú    9 -45.80652 -19.66150     TRUE
46 4044006508 4111   peak  Serra da Picada    9 -46.75626 -18.72909     TRUE
49 4172395838 4111   peak Morro da Chapada   10 -44.08196 -17.74116     TRUE
50 4172395839 4111   peak Morro das Araras   10 -44.21469 -17.86467     TRUE
63 4911005972 4111   peak    Morro do Pião   10 -46.27154 -19.18706     TRUE
```

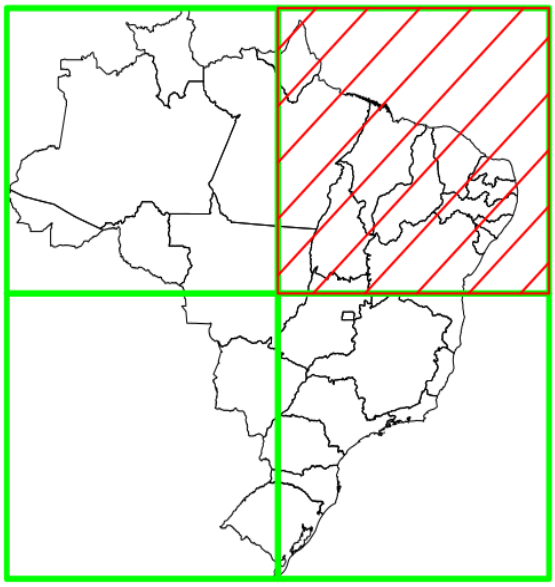
Da mesma maneira que unimos dois data.frame podemos unir um data.frame ao objeto Spatial* usando a função merge:

```
> dfr<-data.frame(codigo=picos$osm_id,valor=runif(length(picos),100,1000))
> picos.merge<-merge(picos,dfr,by.x='osm_id',by.y='codigo')
> head(picos.merge,n=4L)
       osm_id code fclass              name Novo    valor
17  267115440 4111   peak                      8 311.2171
66  559013004 4111   peak    Pico do Itambé    8 604.3856
1  1906457751 4111   peak Morro do Jerônimo    2 786.1517
2  1962488318 4111   peak  Serra da Piedade   10 209.9984
```

2.6.3 - Operações espaciais

Podemos efetuar algumas operações espacias usando o pacote sp. Vamos ver algumas aqui.
Criando um objeto mapa com dois objetos envelopes:

```
> library(raster)
Carregando pacotes exigidos: sp
> library(rgdal)
> estados<-readOGR(dsn='.',layer='estado')
OGR data source with driver: ESRI Shapefile
Source: "/home/andre/livroR", layer: "estados_2010"
with 27 features
It has 5 fields
Integer64 fields read as strings:  id
> #criando um envelope sobre o nosso mapa
> envelope<-raster(estados,nrow=2,ncol=2,vals=1:4)
> names(envelope)<-"Quadrantes"
> envelope<-as(envelope,'SpatialPolygonsDataFrame')
> envelope
class       : SpatialPolygonsDataFrame
features    : 4
extent      : -73.99024, -32.39088, -33.75136, 5.270972  (xmin, xmax, ymin,
ymax)
crs         : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
variables   : 1
names       : Quadrantes
min values  :          1
max values  :          4
> envelope2<-envelope[2,]
> plot(estados)
> plot(envelope,add=TRUE, border='green',lwd=5)
> plot(envelope2,add=TRUE, border='red',lwd=2,density=3,col='red')
```



Vamos apagar a parte do objeto mapa que coincide com o objeto envelope2 usando erase:

```
> apaga<-erase(estados,envelope2)
> # ou podemos usar:
> apaga<-estados-envelope2
> plot(apaga)
```

Ao contrário temos a função intersect que retorna a porção que coincide entre os dois objetos:

```
> intersecta<-intersect(estados,envelope2)
> # ou
> intersect<-estados * envelope2
> plot(intersecta)
```

Podemos usar uma extensão padronizada para extrair uma parte do mapa com extent e crop:

```
> ext<-extent(-50,-42,-17,-15)
> extraido<-crop(estados,ext)
> plot(extraido)
```

2.7 – Manipulação de dados raster

Vamos inicialmente criar um objeto raster vazio com 10 linhas e 10 colunas:

```
> library(raster)
> rst<-raster(ncol=10,nrow=10)
> class(rst)
[1] "RasterLayer"
attr(,"package")
[1] "raster"
>
> rst
class       : RasterLayer
dimensions  : 10, 10, 100  (nrow, ncol, ncell)
resolution  : 36, 18  (x, y)
extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
```

Observamos que o objeto acima possui todos os parâmetros que um geodado precisa: projeção, extensão e resolução de pixel.

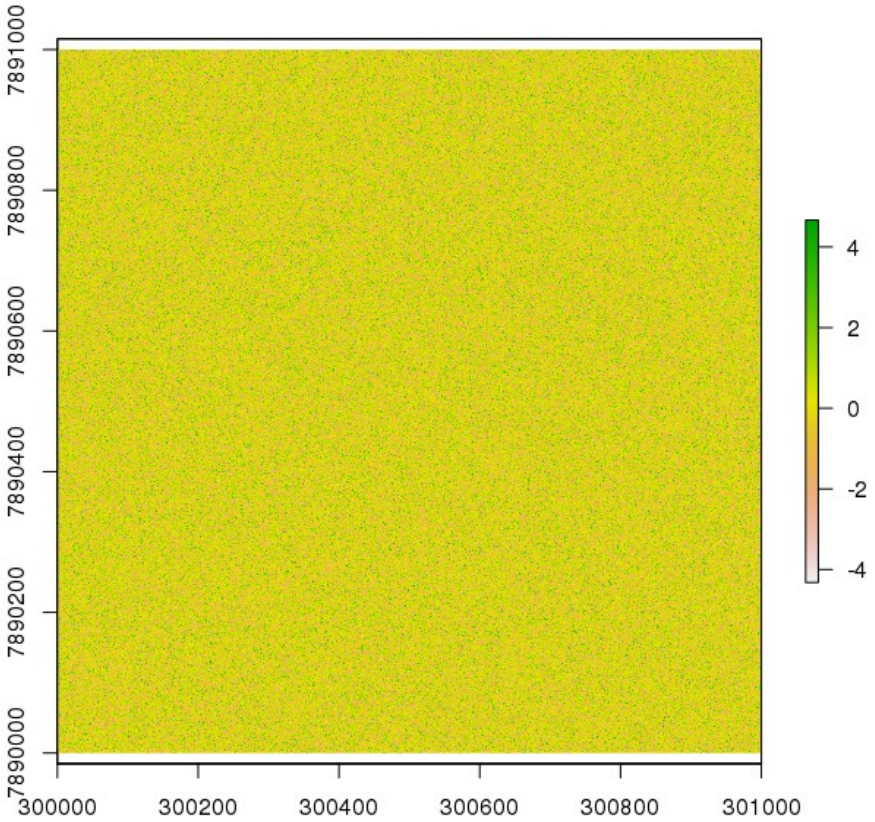
Podemos modificar os parâmetros usando as funções ncol, nrow, res, projection e extent:

```
> projection(rst)<-"+init=epsg:32327"
> extent(rst)<-extent(c(300000,301000,7890000,7891000))
> # mudar resolução mudando numero de linhas e colunas
> ncol(rst)<-1000
> nrow(rst)<-1000
> # ou mudar a resolução diretamente
> res(rst)<-1
>
> rst
class       : RasterLayer
dimensions  : 1000, 1000, 1e+06  (nrow, ncol, ncell)
resolution  : 1, 1  (x, y)
extent      : 3e+05, 301000, 7890000, 7891000  (xmin, xmax, ymin, ymax)
coord. ref. : +init=epsg:32327 +proj=utm +zone=27 +south +ellps=WGS72
+towgs84=0,0,4.5,0,0,0.554,0.2263 +units=m +no_defs
```

Como podemos ver, é muito simples montar uma estrutura de um arquivo raster usando R.

Vamos colocar dados no raster acima criado e plotar ele:

```
> rst[]<-rnorm(n=ncell(rst))
> plot(rst)
```



Podemos usar as funções stack e brick para criar raster com mais de uma banda:

```
> library(raster)
> rst<-raster(ncol=100,nrow=100)
> rst[]<-rnorm(n=ncell(rst))
> stck<-stack(x=c(rst,rst*2,rst*rst))
> brck<-brick(x=c(rst,rst*2,rst*rst))
> stck
class       : RasterStack
dimensions  : 100, 100, 10000, 3  (nrow, ncol, ncell, nlayers)
resolution  : 3.6, 1.8  (x, y)
extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
names       :       layer.1,       layer.2,       layer.3
min values  : -3.972234e+00, -7.944469e+00,  5.943041e-10
max values  :      3.496114,      6.992228,     15.778647

> brck
class       : RasterBrick
dimensions  : 100, 100, 10000, 3  (nrow, ncol, ncell, nlayers)
resolution  : 3.6, 1.8  (x, y)
extent      : -180, 180, -90, 90  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
data source : in memory
names       :       layer.1,       layer.2,       layer.3
min values  : -3.972234e+00, -7.944469e+00,  5.943041e-10
max values  :      3.496114,      6.992228,     15.778647
```

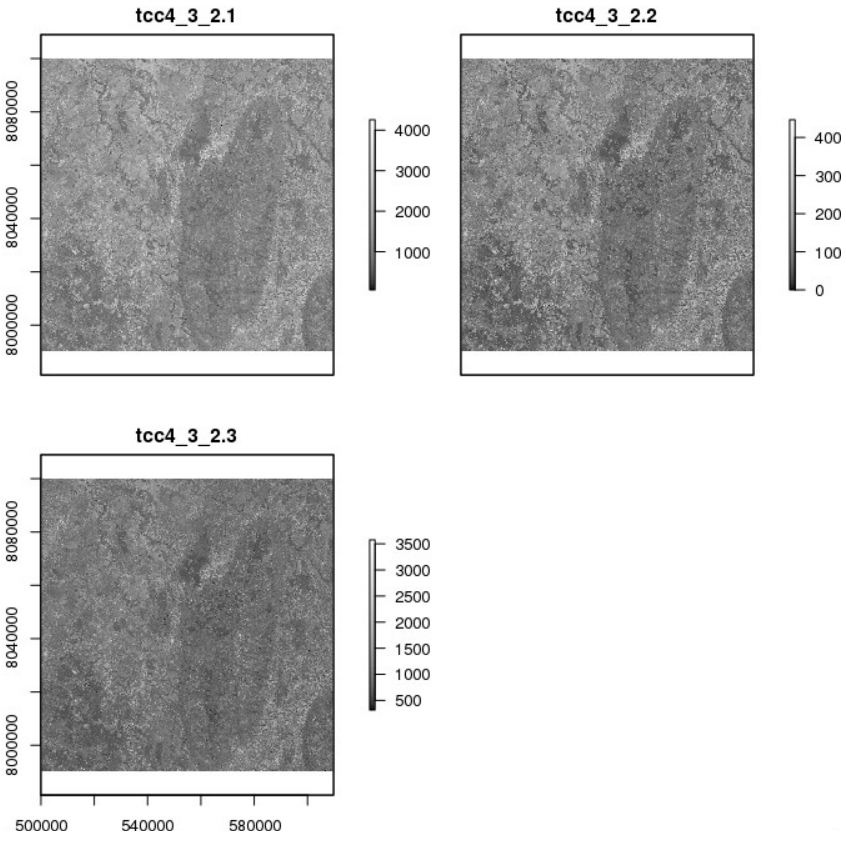
Na grande maioria dos casos, vamos trabalhar com raster já existentes ou criados a partir de dados previamente definidos.

Vamos abrir uma imagem raster do sentinel2 com 3 bandas usando brick e visualizar em RGB :

```
> cabral<-brick('tcc4_3_2.tif')
> plotRGB(cabral,r=1,g=2,b=3,stretch='lin')
```

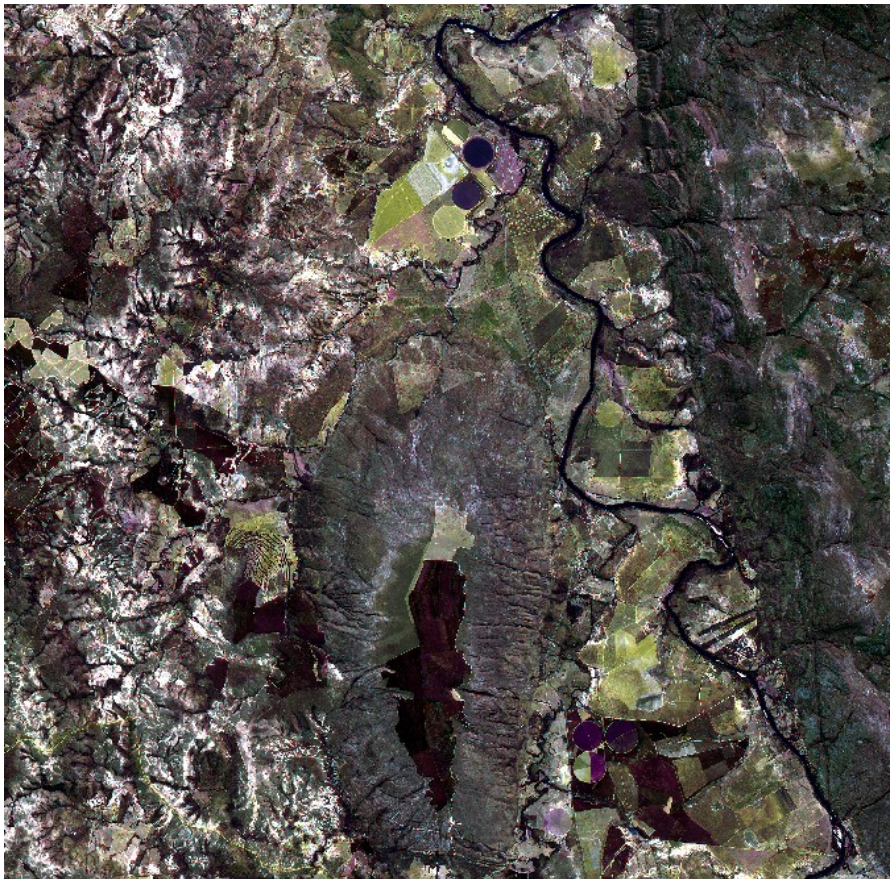
Visualizando as 3 bandas separadamente:

```
> grayscale_colors <- gray.colors(100,start=0.0,end=1.0,gamma=2.2,alpha=NULL)
> plot(cabral,col=grayscale_colors)
```



Podemos extrair uma parte do raster usando a função crop:

```
> e<-extent(c(530000,560000,7990000,8020000))
> cabral.sub<-crop(cabral,e)
> plotRGB(cabral.sub,r=1,g=2,b=3,stretch='lin')
> cabral.sub
class       : RasterBrick
dimensions  : 1488, 1500, 2232000, 3  (nrow, ncol, ncell, nlayers)
resolution  : 20, 20  (x, y)
extent      : 530000, 560000, 7990240, 8020000  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=utm +zone=23 +south +datum=WGS84 +units=m +no_defs
+ellps=WGS84 +towgs84=0,0,0
data source : in memory
names       : tcc4_3_2.1, tcc4_3_2.2, tcc4_3_2.3
min values  :          0,          0,        226
max values  :       4858,       6029,       5235
```



2.7.1 - Álgebra com raster

Podemos efetuar diversas operações algébricas com rasters usando várias funções (abs, round, ceiling, floor, trunc, sqrt, log, log10, exp, cos, sin, atan, tan, max, min, range, prod, sum, any, all) e operadores lógicos tais como: >, >=, <, ==, !.

Podemos efetuar operações desde que o primeiro argumento seja o objeto raster. Vamos criar um raster de 4 linhas e quatro colunas para demonstrar algumas operações:

```
> library(raster)
>  rst<-raster(ncol=4,nrow=4)
> values(rst)<-10:25
>  getValues(rst)
 [1] 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
```

```
> s<-sqrt(rst)
> getValues(s)
 [1] 3.162278 3.316625 3.464102 3.605551 3.741657 3.872983 4.000000 4.123106
 [9] 4.242641 4.358899 4.472136 4.582576 4.690416 4.795832 4.898979 5.000000
> ss<-round(s)+4
> getValues(ss)
 [1] 7 7 7 8 8 8 8 8 8 8 9 9 9 9 9
> sst<-ss/rst
> getValues(sst)
 [1] 0.7000000 0.6363636 0.5833333 0.6153846 0.5714286 0.5333333 0.5000000
 [8] 0.4705882 0.4444444 0.4210526 0.4000000 0.4285714 0.4090909 0.3913043
[15] 0.3750000 0.3600000
> st<-round(sst*10)
> getValues(st)
 [1] 7 6 6 6 6 5 5 5 4 4 4 4 4 4 4 4
> r<-st==4
> getValues(r)
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
[13]  TRUE  TRUE  TRUE  TRUE
> w<-sin(st)
> getValues(w)
 [1]  0.6569866 -0.2794155 -0.2794155 -0.2794155 -0.2794155 -0.9589243
 [7] -0.9589243 -0.9589243 -0.7568025 -0.7568025 -0.7568025 -0.7568025
[13] -0.7568025 -0.7568025 -0.7568025 -0.7568025
```

Vamos mostrar uma aplicação prática de operação com raster. Vamos calcular o NVDI (Índice da diferença de vegetação normalizado) que pode ser descrito por (para dados Sentinel2) como:

(banda8-banda4)/(banda8+banda4)

```
> b4<-raster('b4lr.tif')
> b8<-raster('b8.tif')
> ndvi<-(b8-b4)/(b8+b4)
```

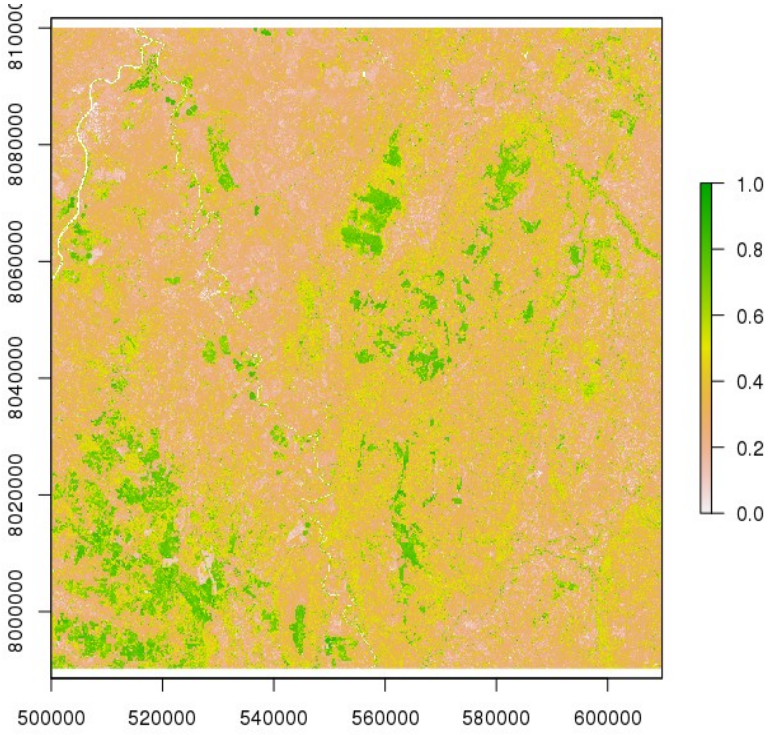
Em vez de usarmos a operação acima, vamos usar a função overlay. A principal razão é que quando trabalhamos com arquivos raster muito grandes a computação de operações podem exaurir a memória do computador. Usando overlay a memória é usada de maneira mais eficiente ***:***

```
> library(raster)
> b4<-raster('b4.tif')
> b8<-raster('b8.tif')
> ndvOver<-function(x,y){
+ ndvi<-(y-x)/(x+y)
+ return(ndvi)
+ }
> ndvi<-overlay(x=b4,y=b8,fun=ndvOver)
```

A função calc age da mesma maneira mas com calc só temos um parâmetro de entrada que pode ser um Layer, Stack ou Brick e o retorno é sempre um objeto Layer. Melhor usar overlay para raster muito grandes, passando camada por camada (layer) como argumento.

Plotando o NDVI:

```
> plot(ndvi,zlim=c(0,1))
```

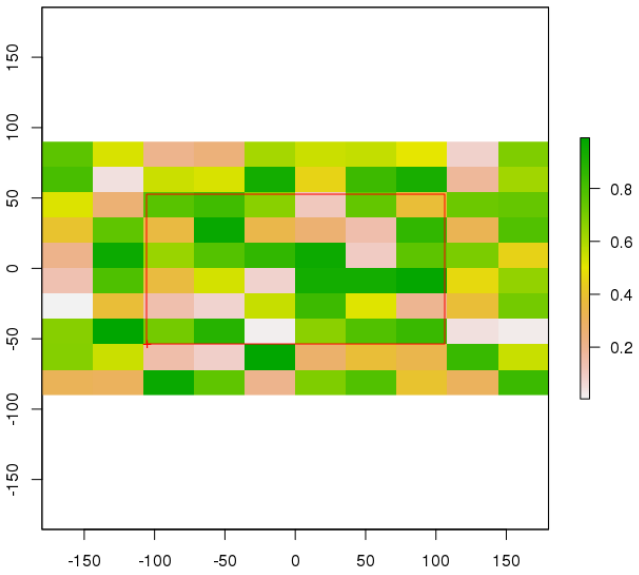


2.7.2 - Funções de alto nível

Várias funções de alto nível são implementadas para objetos RasterLayer. Já vimos acima algumas delas (crop, overlay e calc). Vamos ver algora outras funções de alto nível.

Vamos ver algumas funções que modificam a extensão espacial de um objeto raster. Já vimos que podemos recortar com a função crop fornecendo uma extensão usando extent, podemos também usar a função drawExtent para extrair visualmente na imagem está extensão, use os comandos abaixo e após executar drawExtent, clique em dois pontos no mapa para extrair a extensão:

```
> library(raster)
> r<-raster(ncol=10,nrow=10)
> r[]<-runif(ncell(r))
> plot(r)
> e<-drawExtent()
```

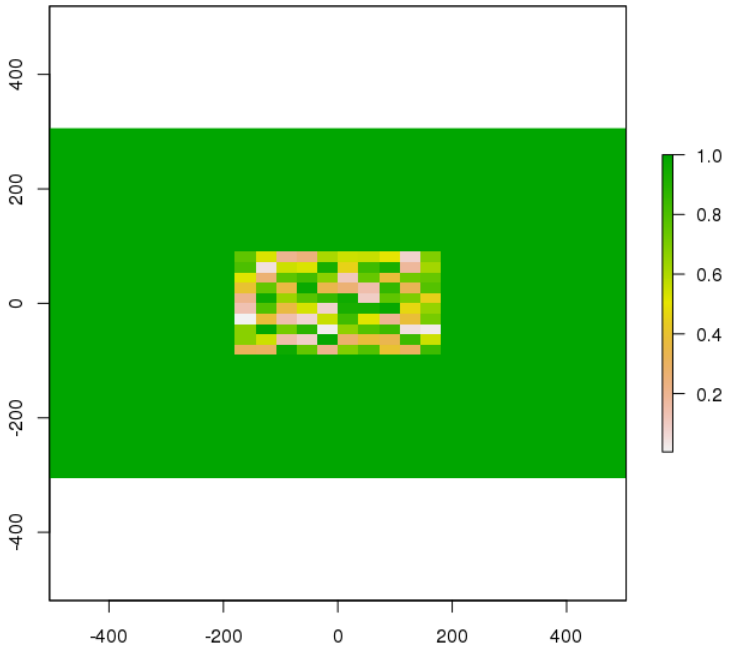


E chamando o valor da extensão teremos algo do tipo:

```
> e
class       : Extent
xmin        : -105.7229
xmax        : 106.2016
ymin        : -53.6344
ymax        : 52.72145
```

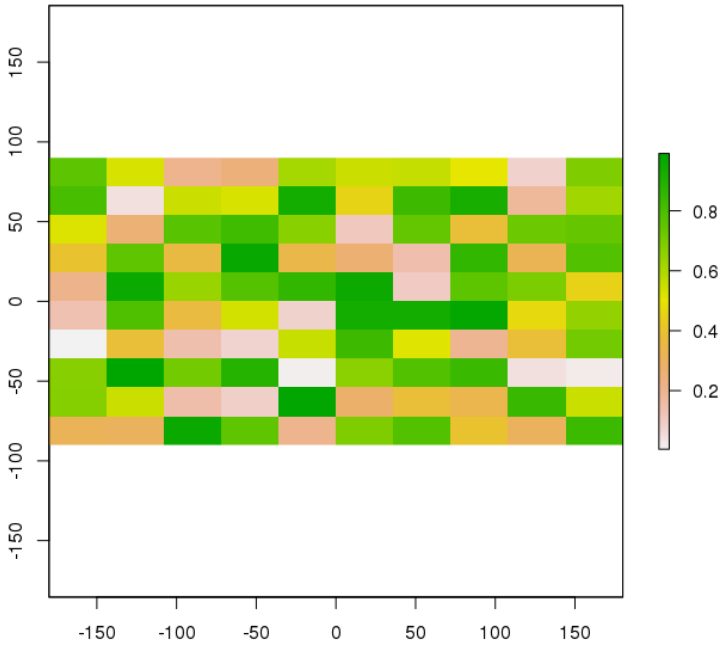
A função extend aumenta a dimensão do objeto raster passado no primeiro argumento com as dimensões fornecidas pelo objeto extent do segundo argumento e atribui o valor do terceiro argumento às novas células criadas:

```
> rex<-extend(r,extent(-500,500,-300,300),value=1)
> rex
class       : RasterLayer
dimensions  : 34, 28, 952  (nrow, ncol, ncell)
resolution  : 36, 18  (x, y)
extent      : -504, 504, -306, 306  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
data source : in memory
names       : layer
values      : 0.004385632, 0.9913201  (min, max)

> plot(rex)
```



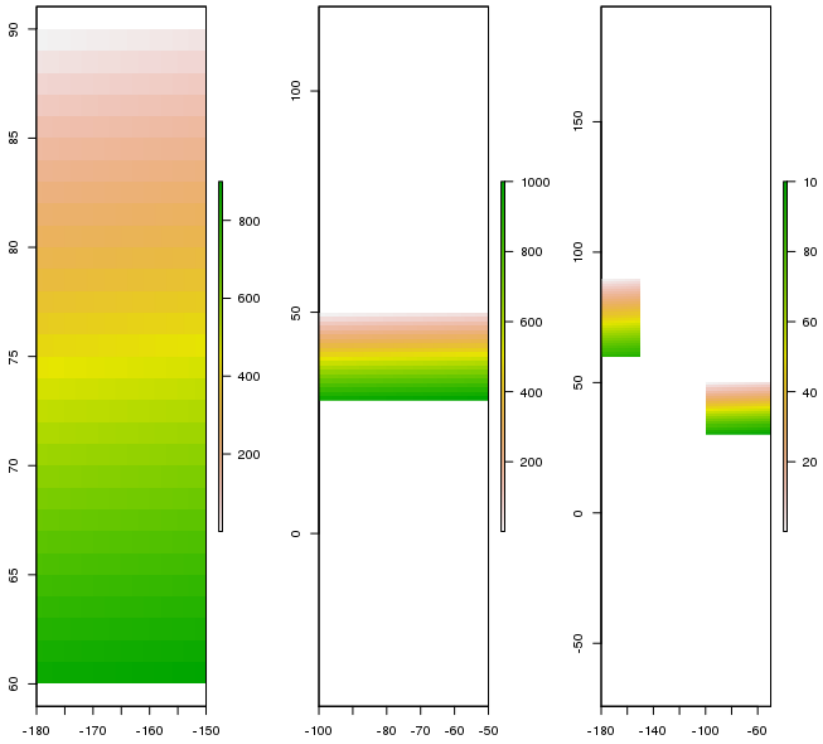
E a função trim remove as linhas e colunas externas do objeto raster do primeiro argumento com os valores fornecidos pelo argumento value:

```
> rtr<-trim(rex,value=1)
> plot(rtr)
```



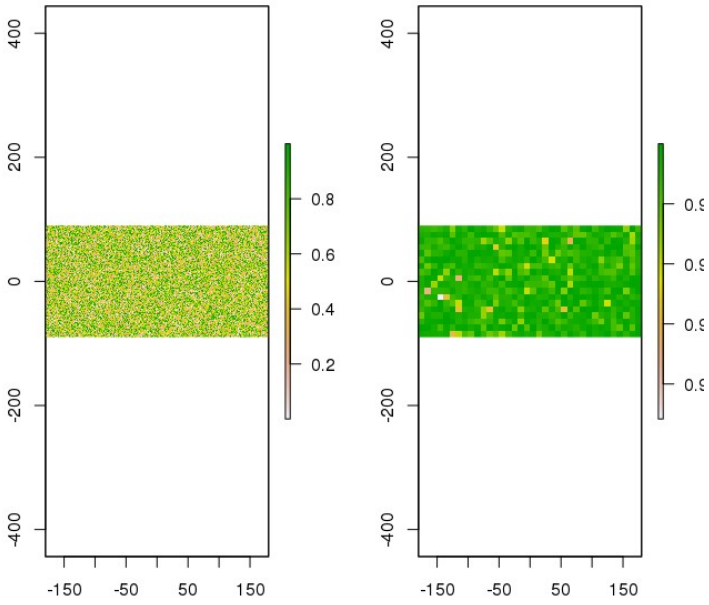
A função merge une dois objetos raster de mesma resolução num único objeto. Se existe sobreposição, o valor da objeto na ordem dos argumentos tem prioridade mas NA é ignorado:

```
> r1<-raster(xmx=-150, ymn=60,ncols=30,nrows=30)
> r1[]<-1:ncell(r1)
> rs<-raster(xmn=-100,xmx=-50,ymx=50,ymn=30)
> res(rs)<-c(xres(r1),yres(r1))
> rs[]<-1:ncell(rs)
> rm<-merge(r1,rs)
> r1
class       : RasterLayer
dimensions  : 30, 30, 900  (nrow, ncol, ncell)
resolution  : 1, 1  (x, y)
extent      : -180, -150, 60, 90  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
data source : in memory
names       : layer
values      : 1, 900  (min, max)
> rs
class       : RasterLayer
dimensions  : 20, 50, 1000  (nrow, ncol, ncell)
resolution  : 1, 1  (x, y)
extent      : -100, -50, 30, 50  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
data source : in memory
names       : layer
values      : 1, 1000  (min, max)
> rm
class       : RasterLayer
dimensions  : 60, 130, 7800  (nrow, ncol, ncell)
resolution  : 1, 1  (x, y)
extent      : -180, -50, 30, 90  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +ellps=WGS84 +towgs84=0,0,0
data source : in memory
names       : layer
values      : 1, 1000  (min, max)
> par(mfrow=c(1,3))
> plot(r1)
> plot(rs)
> plot(rm)
```



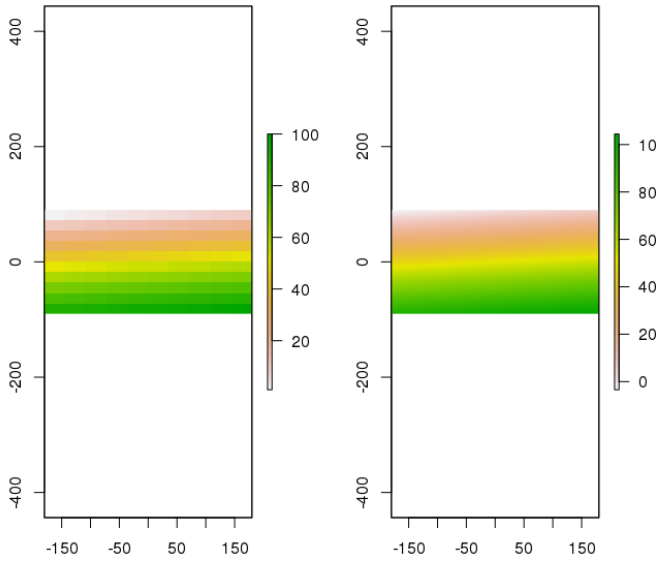
As funções aggregate e disaggregate criam objetos raster com menor ou maior resolução (células maiores ou menores) respectivamente:

```
> r<-raster()
> ra<-aggregate(r,fact=10)
> r<-setValues(r,runif(ncell(r)))
> ra<-aggregate(r,fact=10,fun=max)
> par(mfrow=c(1,2))
> plot(r)
> plot(ra)
```



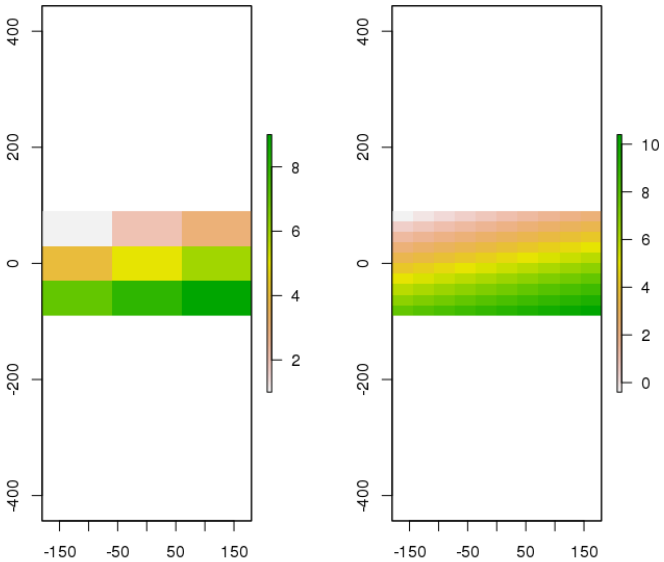

```
> r<-raster(ncol=10,nrow=10)
> rd<-disaggregate(r,fact=c(10,5))
> r[]<-1:ncell(r)
```

```
> rd<-disaggregate(r,fact=c(10,5),method='bilinear')
> par(mfrow=c(1,2))
> plot(r)
> plot(rd)
> dev.off()
```



Podemos usar a função resample para transferir valores entre objetos raster com origem e resoluções diferentes:

```
> r<-raster(nrow=3,ncol=3)
> r[]<-1:ncell(r)
> s<-raster(nrow=10,ncol=10)
> s<-resample(r,s,method='bilinear')
> par(mfrow=c(1,2))
> plot(r)
> plot(s)
```



A função reclassify modifica os valores do objeto raster em faixas de reclassificação fornecidas na matrix do segundo argumento. Essa matrix deve possuir 3 números que correspondem a ‘de’, ‘até’ e ‘valor’ (que será usado para substituir):

```
> r<-raster(ncol=3,nrow=3)
> r[]<-runif(ncell(r))
> as.matrix(r)
          [,1]      [,2]      [,3]
[1,] 0.9192614 0.4917042 0.7887986
[2,] 0.7908096 0.8552953 0.3390543
[3,] 0.3680762 0.5642571 0.6507905
> mtx<-matrix(c(0,0.25,1,0.25,0.5,2,0.5,1,3),ncol=3,byrow=TRUE)
> rcl<-reclassify(r,mtx)
> as.matrix(rcl)
     [,1] [,2] [,3]
[1,]    3    2    3
[2,]    3    3    2
[3,]    2    3    3
```

A função cut é usada para classificar em níveis os valores de um objeto raster de acordo com o intervalo que esses valores estão. Os intervalos são definidos pelo argumento breaks sendo o nível um associado ao valor da esquerda até o nível definido pelo valor da direita (1,2,3...):

```
> r<-raster(ncol=3,nrow=3)
> r[]<-rnorm(ncell(r))*5
> as.matrix(r)
           [,1]      [,2]       [,3]
[1,]  0.7246631 -2.543486  0.8648624
[2,]  0.1413446 -6.353755 -5.3065005
[3,] -7.2165514  6.167294 -7.9453636
> brks<--2:2*2
> rc<-cut(r,breaks=brks)
> as.matrix(rc)
     [,1] [,2] [,3]
[1,]    3    1    3
[2,]    3   NA   NA
[3,]   NA   NA   NA
```

A função subs substitui valores em um objeto raster* (primeiro argumento) com valores de um data.frame (segundo argumento). O data.frame deve ter uma coluna para identificar a chave (ID) que deve coincidir com os valores das células do objeto raster, e uma ou mais colunas com os valores de substituição. Por padrão a primeira e a segunda coluna são usadas mas podemos especificar quais colunas com os argumentos by e which:

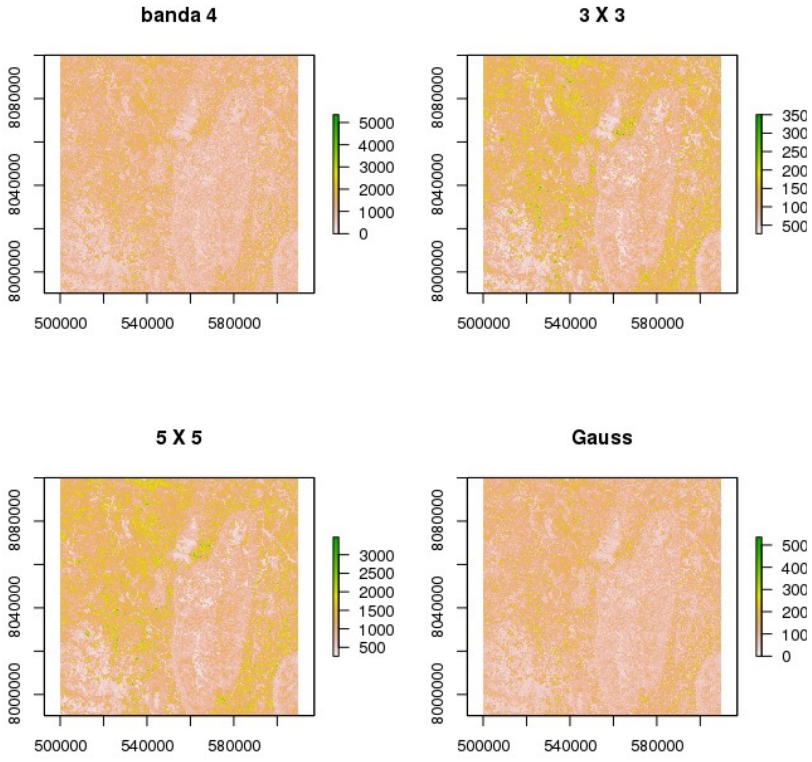
```
> r<-raster(ncol=3,nrow=3)
> r[]<-round(runif(ncell(r))*10)
> df<-data.frame(id=2:8, v=c(10,10,11,11,12,13,14))
> df
  id  v
1  2 10
2  3 10
3  4 11
4  5 11
5  6 12
6  7 13
7  8 14
> as.matrix(r)
     [,1] [,2] [,3]
[1,]    4    3    8
[2,]    4    7    2
[3,]    4    1    5
```

```
> x<-subs(r,df)
> as.matrix(x)
     [,1] [,2] [,3]
[1,]   11   10   14
[2,]   11   13   10
[3,]   11   NA   11
> x<-subs(r,df,subsWithNA=FALSE)
> as.matrix(x)
     [,1] [,2] [,3]
[1,]   11   10   14
[2,]   11   13   10
[3,]   11    1   11
```

Vamos dar continuidade com as funções de alto nível falando um pouco agora sobre funções que filtram e apresentam resultados espaciais dos objetos Raster*.

A função focal funciona como um filtro (janela móvel) usando dados das células vizinhas e uma matriz de pesos ou uma combinação com uma função. Vamos aplicar 3 diferentes exemplos abaixo, um filtro de janela 3x3, um 5x5 e um gaussiano. Está função só funciona com objetos RasterLayer :

```
> b4<-raster('b4lr.tif')
> r3<-focal(b4,w=matrix(1/9,nrow=3,ncol=3))
> r5<-focal(b4,w=matrix(1/25,nrow=5,ncol=5))
> gf<-focalWeight(b4,2,"Gauss")
> rg<-focal(b4,w=gf)
> par(mfrow=c(2,2))
> plot(b4,main='banda 4')
> plot(r3,main='3 X 3')
> plot(r5,main='5 X 5')
> plot(rg,main='Gauss')
```



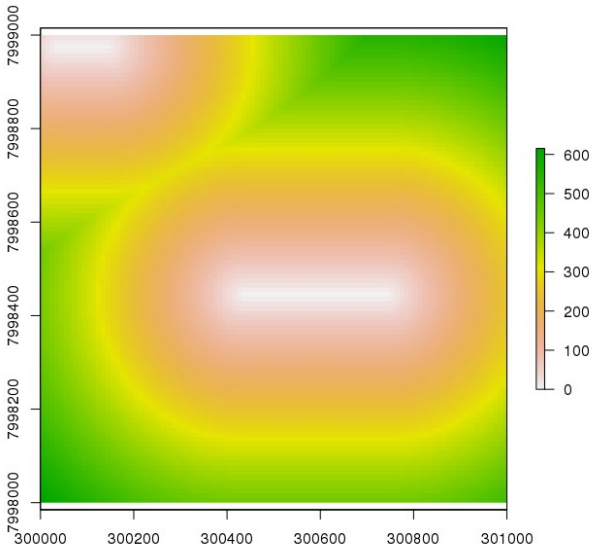
As funções relacionadas com distância são muito importantes na análise espacial de dados geológicos. Vamos aqui mostrar por alto algumas delas (voltaremos a falar mais delas na parte 2).

Para demostrar estas funções vamos criar o seguinte objeto RasterLayer :

```
> rl<-raster(ncol=100,nrow=100,xmx=301000,xmn=300000,ymn=7998000,ymx=7999000)
> crs(rl)<-CRS('+init=epsg:32723')
> rl[]<-NA
> rl[3,4:15]<-34
> rl[56,44:75]<-18
```

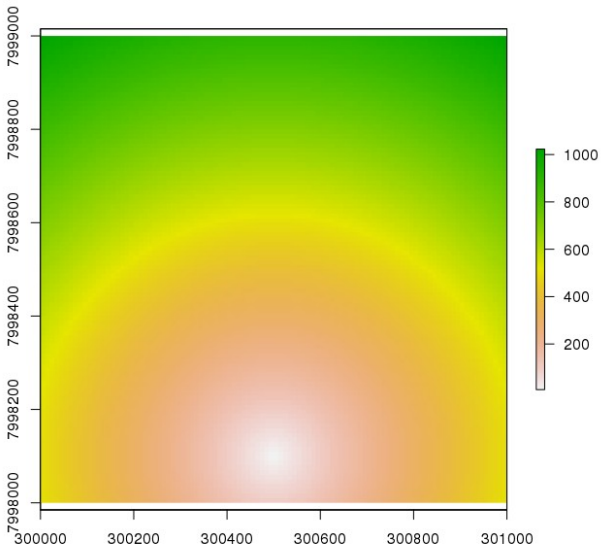
A função distance calcula todos os valores da célula até o a próxima célula que não tenha o valor NA. O resultando é objeto RasterLayer também:

```
> distance(rl)
> plot(dist)
```



A função distanceFromPoints calcula a distância de todas as células do objeto RasterLayer* com relação ao ponto(s) dado:

```
> xy<-c(300500,7998100)
> dist<-distanceFromPoints(rl,xy)
> plot(dist)
```

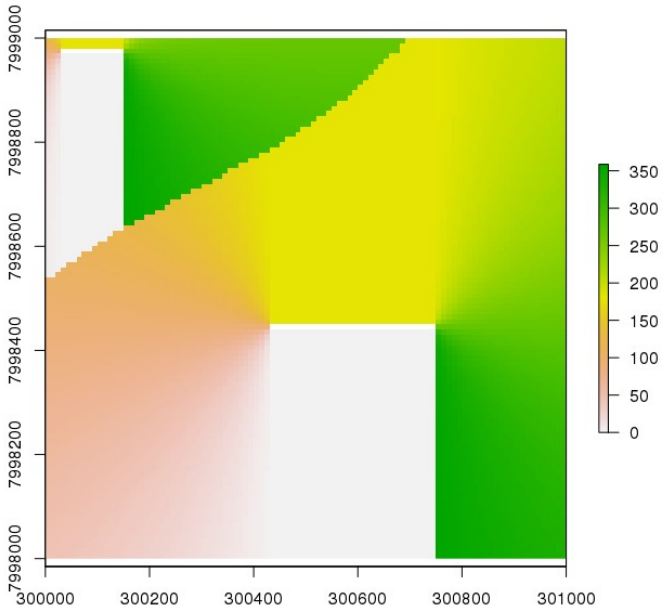


A função pointDistance retorna a distância entre os pontos informados:

```
> xy<-c(300500,7998100)
> xy2<-c(300001,7998999)
> pointDistance(xy,xy2,lonlat=FALSE)
[1] 1028.203
```

A função direction calcula a direção para a próxima célula que o valor não é NA. O resultado é um objeto RasterLayer:

```
> dir<-direction(rl,degrees=TRUE)
> plot(dir)
```



2.7.3 - Funções informativas

Podemos usar funções de estatísticas de células para obtermos um sumário de todas as células em um objeto raster. Abaixo vamos ver como podemos usar a função cellStats para saber informações do conjunto de células; freq para fazer uma tabela de frequência ou para contar o número de células com um valor específico; zonal para sumarizar um objeto raster usando zonas definidas; e crosstab para fazermos tabulação cruzada entre dois objetos RasterLayer:

```
> r<-raster(ncol=36,nrow=18)
> r[]<-runif(ncell(r))
> cellStats(r,mean)
[1] 0.4956946
> s<-r
> s[]<-round(runif(ncell(r))*5)
> zonal(r,s,'mean')
     zone      mean
[1,]    0 0.5195020
[2,]    1 0.4863886
[3,]    2 0.4928189
[4,]    3 0.5067917
[5,]    4 0.4826296
[6,]    5 0.5002245
> freq(s)
     value count
[1,]     0    62
[2,]     1   127
[3,]     2   134
```

```
[4,]     3   138
[5,]     4   130
[6,]     5    57
> freq(s,value=4)
[1] 130
> ctb<-crosstab(r*3,s)
> head(ctb)
  Var1 Var2 Freq
1    0    0    9
2    1    0   20
3    2    0   21
4    3    0   12
5 <NA>    0    0
6    0    1   25
```

2.7.4 - Funções de auxílio

O número da célula é um conceito importante no pacote raster. Os dados de um raster podem ser pensados como uma matrix mas na realidade um objeto RasterLayer é mais comumente tratado como um vector. As células são numeradas da célula superior esquerda e continuam no lado esquerdo na próxima linha e terminam no lado direito da última linha (célula inferior direita). Existem várias funções de auxílio para determinarmos qual coluna e linha uma célula ocupa e vice-versa e também para sabermos as coordenadas x e y de determinada célula:

```
> rl<-raster(ncol=100,nrow=100,xmx=301000,xmn=300000,ymn=7998000,ymx=7999000)
> crs(rl)<-CRS('+init=epsg:32723')
> rl
class       : RasterLayer
dimensions  : 100, 100, 10000  (nrow, ncol, ncell)
resolution  : 10, 10  (x, y)
extent      : 3e+05, 301000, 7998000, 7999000  (xmin, xmax, ymin, ymax)
coord. ref. : +init=epsg:32723 +proj=utm +zone=23 +south +datum=WGS84 +units=m
+no_defs +ellps=WGS84 +towgs84=0,0,0
> ncol(rl)
[1] 100
> nrow(rl)
[1] 100
> ncell(rl)
[1] 10000
> rowFromCell(rl,3490)
[1] 35
> colFromCell(rl,3490)
[1] 90
> cellFromRowCol(rl,34,67)
[1] 3367
> xyFromCell(rl,3490)
          x       y
[1,] 300895 7998655
> cellFromXY(rl,c(300456,7998500))
[1] 5046
> colFromX(rl,300456)
[1] 46
> rowFromY(rl,7998456)
[1] 55
```

2.7.5 - Acessando Valores das células

Podemos ter acesso aos valores de células usando vários métodos. Use getValues para ver todos os valores em uma única linha e getValuesBlock para ler um bloco (retângulo) de células:

```
> b4<- raster('brlr.tif')
> v<-getValues(b4,50)
> v[100:110]
 [1] 1180 1174 1239 1328 1572 1558 1235 1446 1302 1435 1371
> getValuesBlock(b4,50,3,51,3)
 [1] 1006  868  885  858 1013 1007  985  870  772
```

Podemos também ler valores usando o número das células ou coordenadas destas usando a função extract:

```
> celulas<-cellFromRowCol(b4,50,35:40)
> celulas
[1] 269045 269046 269047 269048 269049 269050
> extract(b4,celulas)
[1]  640 1573 1133  895 1067  703
> xy<-xyFromCell(b4,celulas)
> xy
          x       y
[1,] 500670 8099050
[2,] 500690 8099050
[3,] 500710 8099050
[4,] 500730 8099050
[5,] 500750 8099050
[6,] 500770 8099050
> extract(b4,xy)
[1]  640 1573 1133  895 1067  703
```

Podemos usar extract também usando um objeto SpatialPolygons* ou SpatialLines* como o segundo argumento, nesses casos, células que forem tocadas pela linha ou dentro do polígono serão mostradas.

Embora não aconselhável para raster muito grandes, podemos acessar valores de células usando índices:

```
> b4[celulas]
[1]  640 1573 1133  895 1067  703
> b4[50,40]

703
> b4[2,2]

792
> b4[296459]

1256
```

Terminamos aqui a abordagem sobre a manipulação de dados espaciais com R. O próximo passo é aprender sobre a análise e geoprocessamento de dados espaciais.

PARTE 3 - Análise Espacial

3.1 – Análise de Dados Pontuais e gerando dados poligonais

Para essa parte usaremos dados de geoquímica de solo extraído de https://pubs.usgs.gov/ds/801/ modificado para reduzir o número de colunas. Trata-se de análise química do horizonte C espalhados por todo os Estados Unidos continental, ***Smith et all. 2013*** (agradecimento ao USGS).

3.1.1 - Visualizando e inspecionando dados pontuais

Vamos carregar os dados e criar um objeto SpatialPointsDataFrame com eles:

```
> geoqui<-read.csv('usa_quim_solo.csv')
> library(raster)
> coordinates(geoqui)<-~Longitude+Latitude
> head(geoqui,n=1)
       ponto Ag.ppm Al.pct As.ppm Ba.ppm Be.ppm Bi.ppm Ca.pct Cd.ppm Ce.ppm
P_0001 P_0001      1   5.42    4.8    121    0.6   0.24   0.02    0.1   36.3
       Co.ppm Cr.ppm Cs.ppm Cu.ppm Fe.pct Ga.ppm Hg.ppm In.ppm K.pct La.ppm
P_0001    3.7     26      5   12.3    3.3   12.5   0.01   0.06  0.35   20.3
       Li.ppm Mg.pct Mn.ppm Mo.ppm Na.pct Nb.ppm Ni.ppm P.ppm Pb.ppm Rb.ppm
P_0001     17   0.11     41   1.32   0.02   11.3   11.4   110     14   25.9
       S.pct Sb.ppm Sc.ppm Se.ppm Sn.ppm Sr.ppm Te.ppm Th.ppm Ti.pct Ti.ppm
P_0001  0.03   0.52    7.5    0.6    1.8   18.8    0.1   10.8    0.4    0.3
       U.ppm V.ppm W.ppm Y.ppm Zn.ppm
P_0001   2.8    68   0.9   7.7     28
```

O dado original está em NAD-83 (SRID 4269). Ao carregarmos o dado ele não possui CRS, então assinalaremos NAD-83 e convertemos para WGS-84. Sempre que possível trabalhe com WGS-84:

```
> t.str <- CRS('+init=epsg:4326')
> crs(geoqui)<-CRS('+init=epsg:4269')
> geoqui.sp<-spTransform(geoqui, t.str)
> crs(geoqui.sp)
CRS arguments:
 +init=epsg:4326 +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84
+towgs84=0,0,0
```

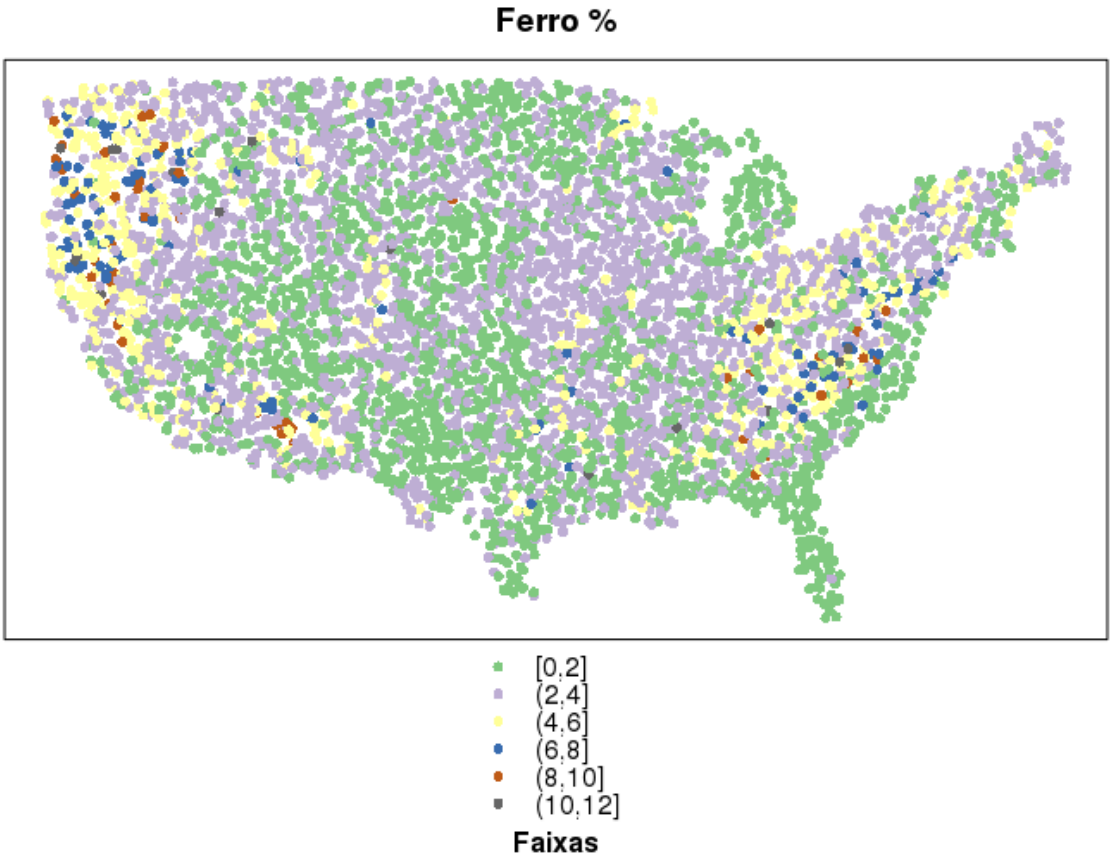
Vamos visualizar alguns dos dados usando spplot. Podemos assim ter uma ideia de como os dados estão distribuídos de uma maneira geral de forma rápida e efetiva antes de prosseguirmos com algum tratamento.

Vamos plotar os valores de Fe em um mapa com sspplot quebrando os dados em 0, 2, 4, 6, 8, 10 e12 % com a diretiva cut, mas antes definiremos uma paleta de cor com o pacote RcolorBrewer:

```
> library(RColorBrewer)
> paleta <- brewer.pal(n=8,name = "Accent")
```

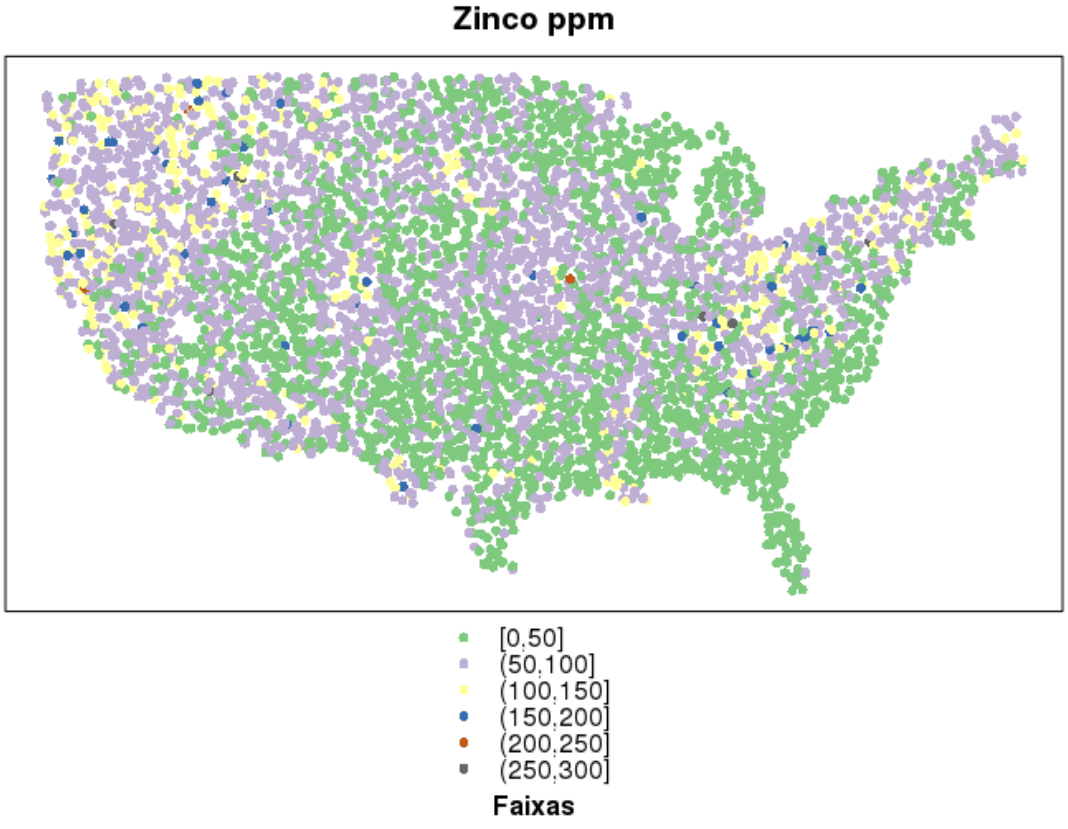
Experimente diferentes cores e diferentes cortes de valores que melhor expresse a distribuição dos dados. Use max e min para conhecer a faixa de valores de cada elemento, use display.brewer.all() para ver combinações de cores a serem usadas. Agora vamos plotar os valores de Fe dos dados:

```
> spplot(geoqui.sp,'Fe.pct',col.regions=paleta,cuts=c(0,2,4,6,8,10,12),
cex=0.75,main='Ferro %', sub='Faixas')
```

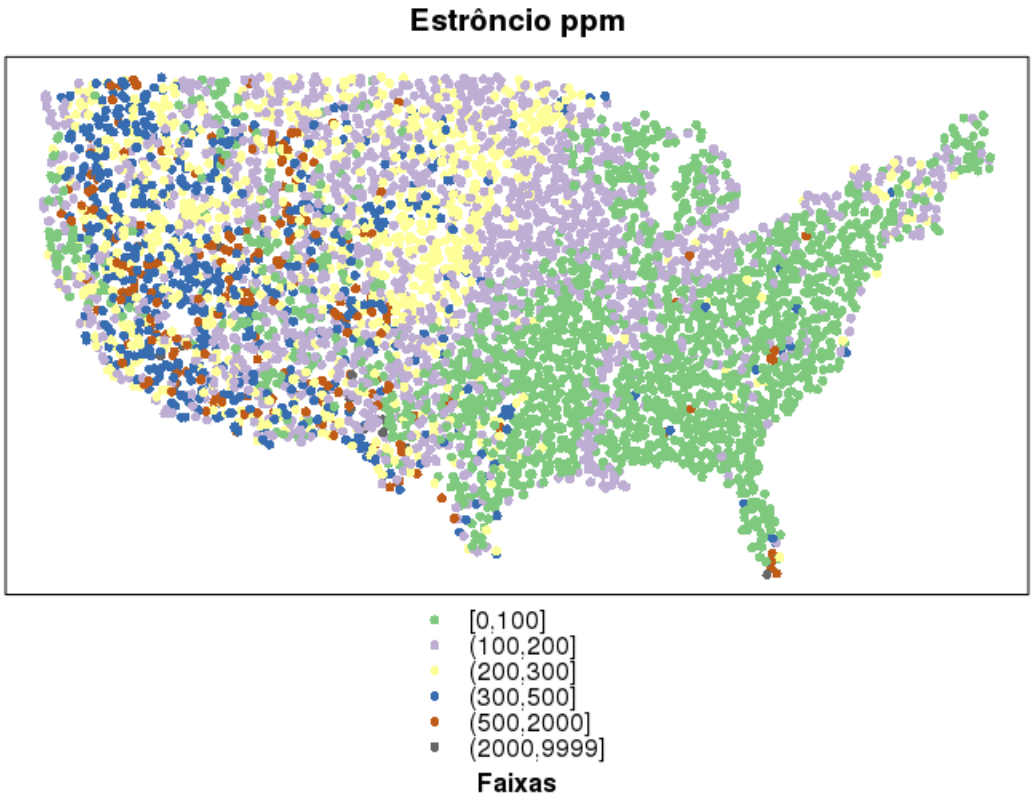


Agora vendo valores para Zinco, Estrôncio, Urânio, e Estanho em diferentes formatos:

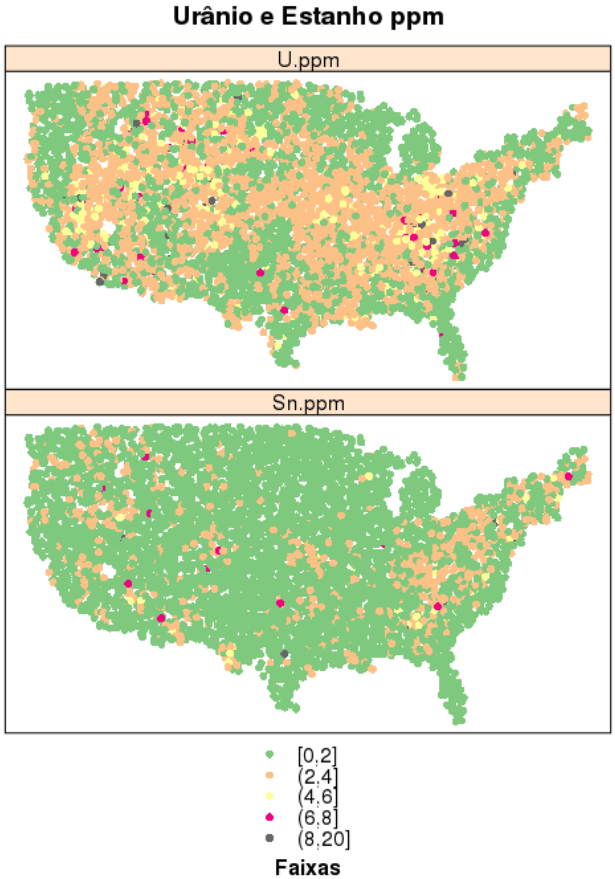
```
> spplot(geoqui.sp,'Zn.ppm',col.regions=paleta,cuts=c(0,50,100,150,200,250),
cex=0.75,main='Zinco ppm', sub='Faixas')
```



```
> spplot(geoqui.sp,'Sr.ppm',col.regions=paleta,cuts=c(0,100,200,300,500,
2000,9999),cex=0.75,main='Estrôncio ppm', sub='Faixas')
```



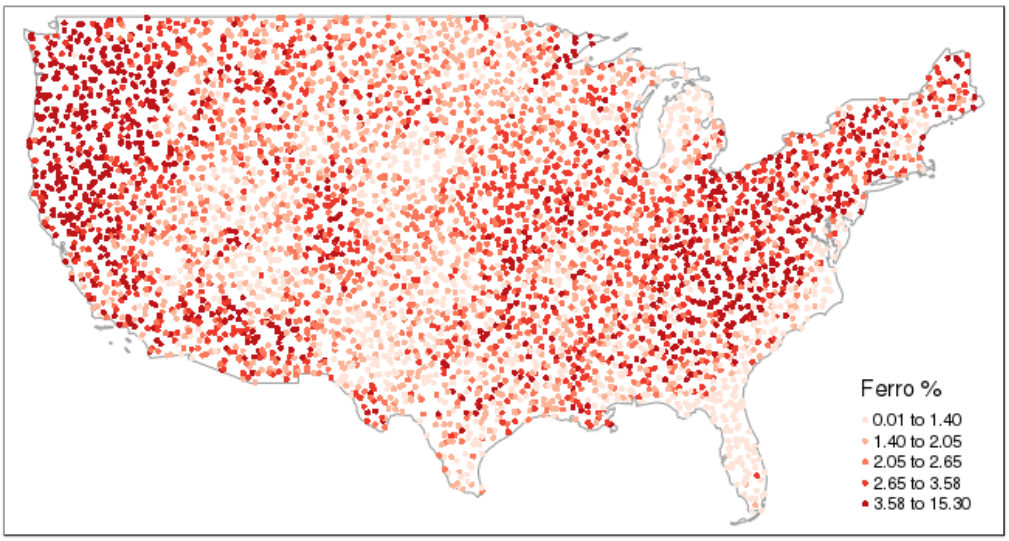

```
> spplot(geoqui.sp,c('Sn.ppm','U.ppm'),col.regions=paleta,cuts=c(0,
2,4,6,8,20),cex=0.75,main='Urânio e Estanho ppm', sub='Faixas')
```



Podemos utilizar o pacote tmap para efetuar a visualização dos dados, qual pacote usar é uma escolha pessoal ou usar uma tarefa específica que as funções de um pacote realizam melhor que as de outro pacote.

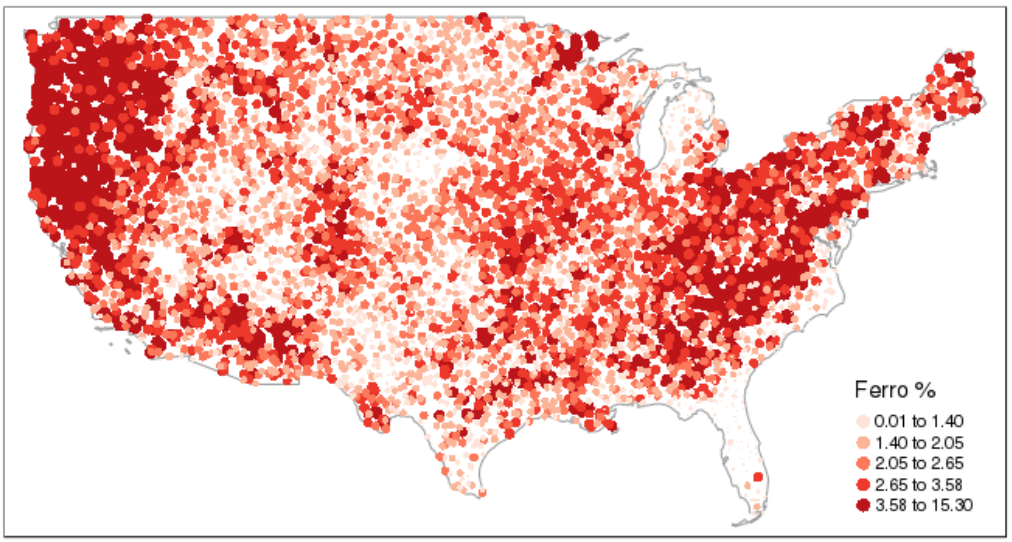
Vamos ver como plotar dados pontuais com o pacote tmap:

```
> geoqui<-read.csv('usa_quim_solo.csv')
> library(raster)
> coordinates(geoqui)<-~Longitude+Latitude
> library(rgdal)
> us<-readOGR(dsn='.',layer='us')
> t.str <- CRS('+init=epsg:4326')
> crs(geoqui)<-CRS('+init=epsg:4269')
> geoqui.sp<-spTransform(geoqui, t.str)
> library(tmap)
> tm_shape(us) + tm_borders(alpha=.4,col='black') +tm_shape(geoqui.sp) +
tm_dots(col = "Fe.pct", palette = "Reds", style = "quantile",scale=2,
title='Zinco ppm')
```



Tamanho do símbolo proporcional ao teor:

```
> tm_shape(us) + tm_borders(alpha=.4,col='black') +tm_shape(geoqui.sp) +
tm_dots(size='Fe.pct',col = "Fe.pct", palette = "Reds", style = "quantile",
title='Zinco ppm',legend.size.show=FALSE)
```

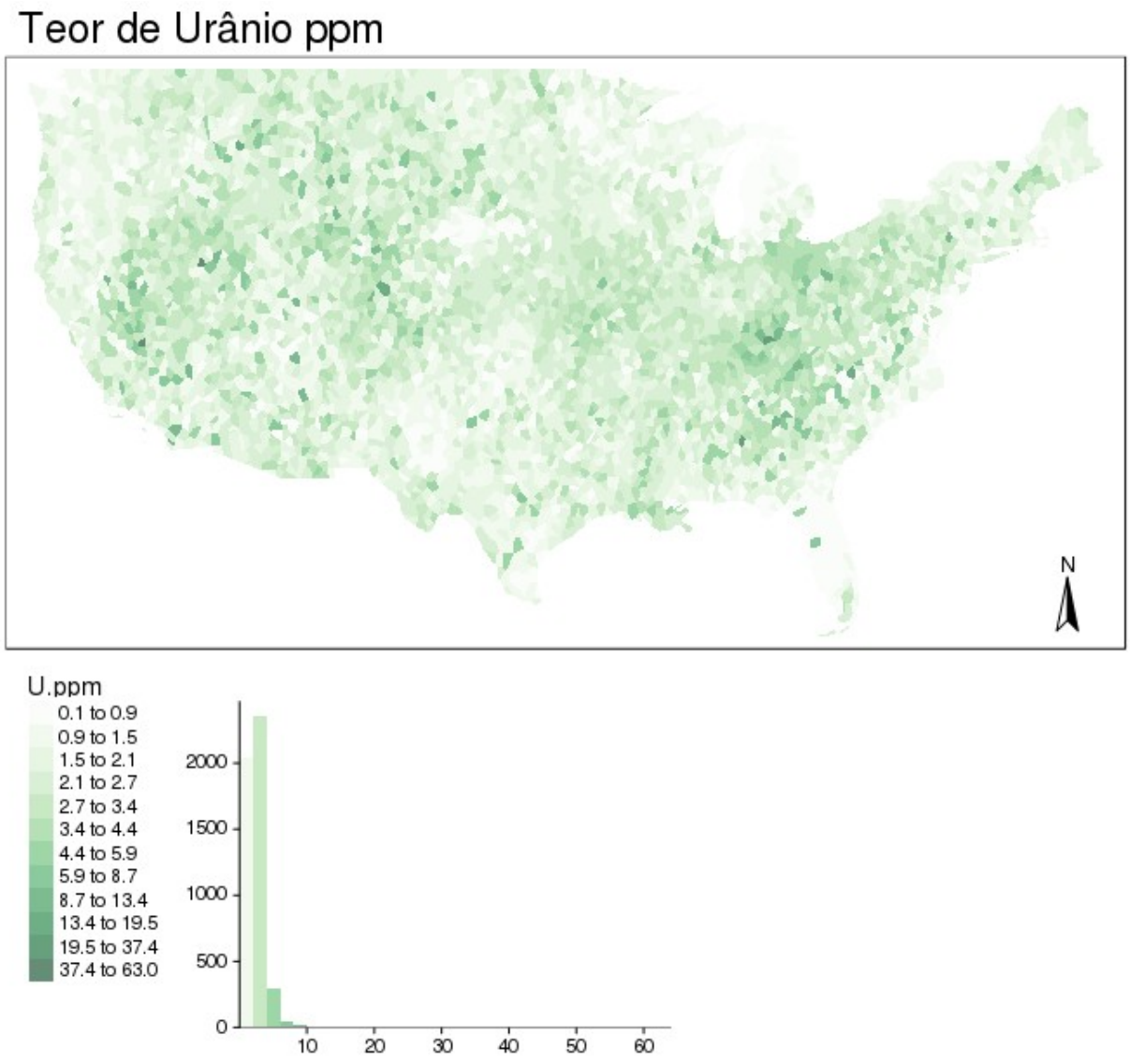


Vamos criar polígonos representando cada um dos pontos amostrados usando a função voronoi do pacote dismo.

```
> library(dismo)
> v<-voronoi(geoqui.sp)
> vor.us<-intersect(v,us)
```

Plotando os polígonos usando teor de Urânio:

```
> tm_shape(vor.us) + tm_fill(alpha=.6,n=12,legend.hist=TRUE,col =
"U.ppm",style="jenks",palette="Greens")+tm_layout(main.title='Teor de Urânio
ppm',legend.outside.position = c("bottom"),legend.outside=TRUE,legend.stack
='horizontal',legend.hist.height=1.05,legend.hist.width=1.05)+tm_compass()
```

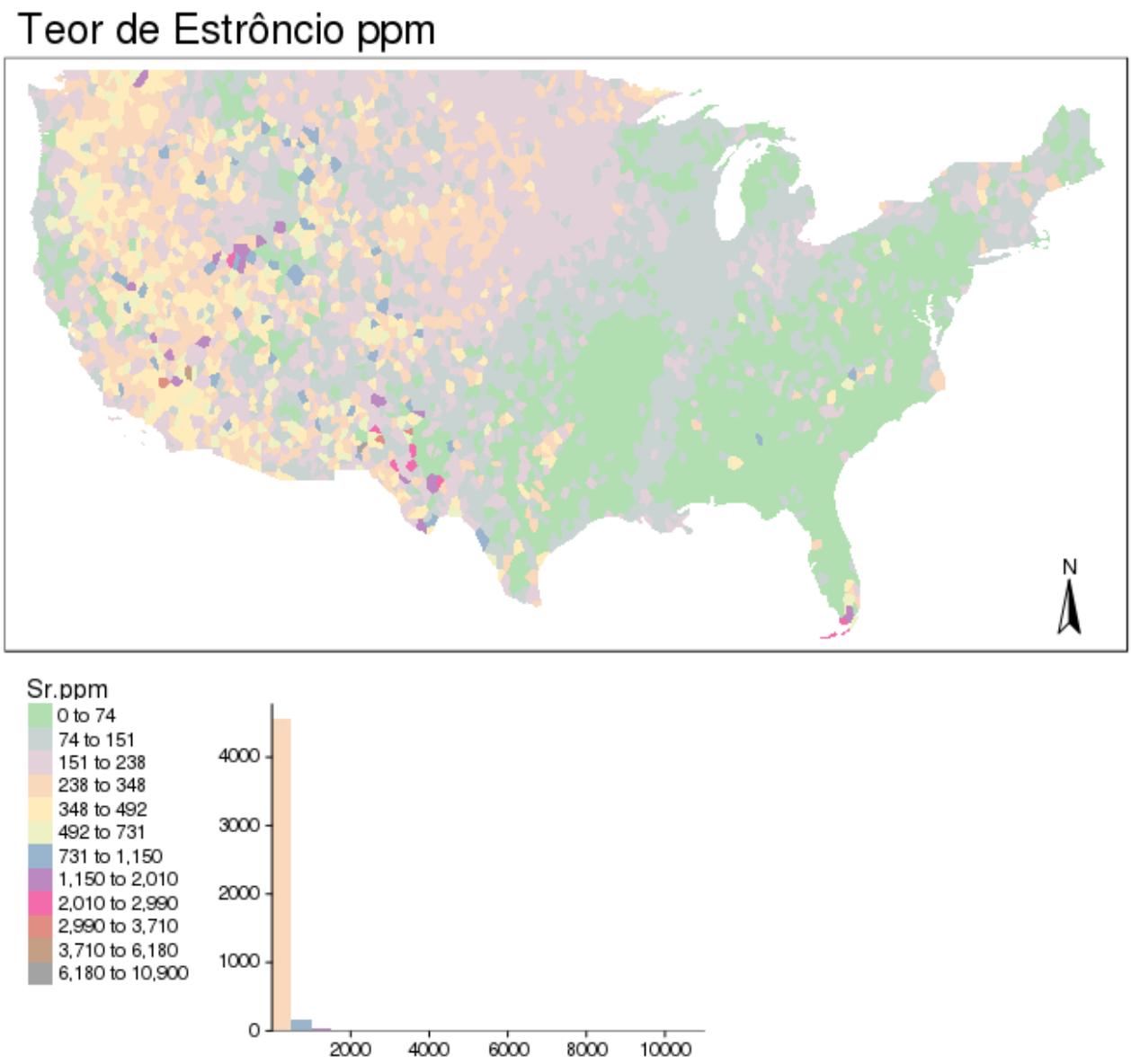


Podemos usar uma paleta de cores do pacote RcolorBrewer da mesma forma feita com sppplot:

```
> library(RColorBrewer)
> paleta <- brewer.pal(n=8,name = "Accent")
```

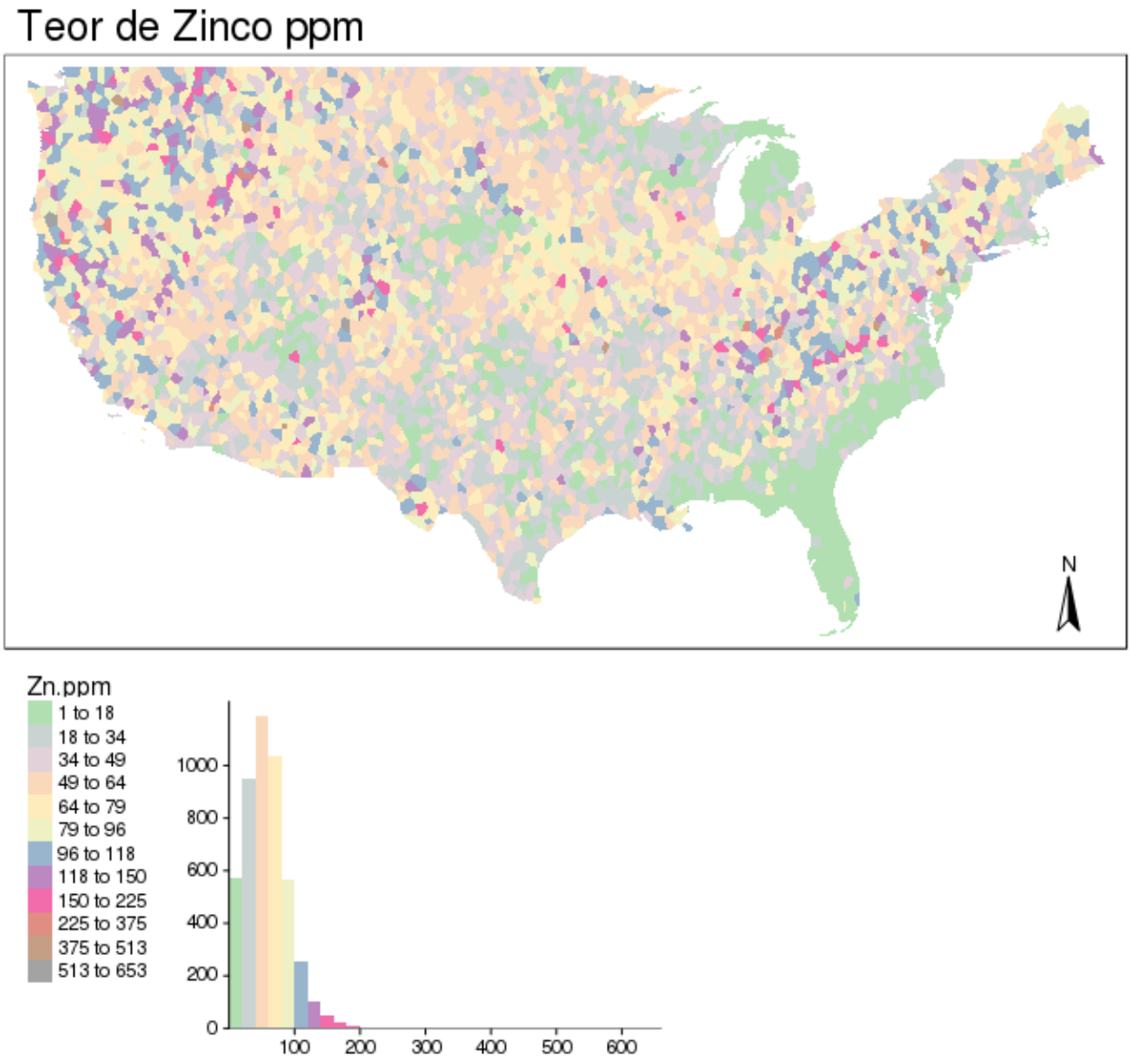
E podemos plotar usando a paleta recém-criada com:

```
> tm_shape(vor.us) + tm_fill(alpha=.6,n=12,legend.hist=TRUE,col =
"Sr.ppm",style="jenks",palette=paleta)+tm_layout(main.title='Teor de Estrôncio
ppm',legend.outside.position = c("bottom"),legend.outside=TRUE,legend.stack=
'horizontal',legend.hist.height=1.05,legend.hist.width=1.05)+tm_compass()
```



E agora vamos plotar o teor de Zinco associado a cada polígono

```
> tm_shape(vor.us) + tm_fill(alpha=.6,n=12,legend.hist=TRUE,col = "Zn.ppm",
style="jenks",palette=paleta)+tm_layout(main.title='Teor de Zinco ppm',
legend.outside.position = c("bottom"),legend.outside=TRUE, legend.stack=
'horizontal', legend.hist.height=1.05,legend.hist.width=1.05)+tm_compass()
```



O pacote tmap oferece uma boa variedade de plotagem de dados seja do tipo ponto ou do tipo polígono. Inclusive apresentando histograma de distribuição dos dados.

Os mapas foram criados usando o parâmetro estilo jenks (style=’jenks’) que demora um pouco para gerar a distribuição de dados nas faixas de cores. Use style=’quantile’ para maior rapidez no resultado mas com uma pior distribuição das cores com relação aos dados.

3.1.2 - Trabalhando com distâncias entre dados pontuais

Muitos dos processos envolvidos em análise espacial, como era de se esperar, envolvem o parâmetro distância.

Carregando os dados:

```
> geoqui<-read.csv('usa_quim_solo.csv')
> library(raster)
> coordinates(geoqui)<-~Longitude+Latitude
> t.str <- CRS('+init=epsg:4326')
> crs(geoqui)<-CRS('+init=epsg:4269')
> geoqui.sp<-spTransform(geoqui, t.str)
```

Para obtermos distância entre pontos usamos a função spDists que toma como parâmetros dois objetos Spatial* (no nosso caso usaremos o mesmo objeto duas vezes) e um terceiro parâmetro que informa se o dado é Euclidiano (ex. UTM) ou Longitude / Latitude. Podemos usar arrays nessa função como os dois primeiros argumentos onde a primeira e segunda colunas denotando Longitude e Latitude e cada linha representa um ponto.
Criando a nossa matriz de distâncias:

```
> distancias<-spDists(geoqui.sp, y = geoqui.sp, longlat = TRUE)
> rownames(distancias)<-geoqui$ponto
> colnames(distancias)<-geoqui$ponto
> distancias[1:6,1:6] #visualizando as 6 primeiras linhas e colunas
         P_0001    P_0002   P_0003    P_0004    P_0005    P_0006   ...
P_0001   0.0000 282.13149 118.3714 198.21403 362.64736 337.98723
P_0002 282.1315   0.00000 307.6084  83.97997 105.25003  67.57494
P_0003 118.3714 307.60840   0.0000 233.78818 406.06826 373.21370
P_0004 198.2140  83.97997 233.7882   0.00000 172.81599 142.61792
P_0005 362.6474 105.25003 406.0683 172.81599   0.00000  39.74719
P_0006 337.9872  67.57494 373.2137 142.61792  39.74719   0.00000
```

Dando prosseguimento vamos ver como podemos trabalhar com essa matriz de distâncias entre pontos para obter outras informações:

Calculando os pontos distantes entre 100 e 200 Km:

```
> # matrix distâncias entre 100 e 200 km
> adj.100.200<-distancias < 200 & distancias>100
> diag(adj.100.200)<-NA #distância dele com ele mesmo não se aplica
> adj.100.200[1:6,1:6]
       P_0001 P_0002 P_0003 P_0004 P_0005 P_0006   ...
P_0001     NA  FALSE   TRUE   TRUE  FALSE  FALSE
P_0002  FALSE     NA  FALSE  FALSE   TRUE  FALSE
P_0003   TRUE  FALSE     NA  FALSE  FALSE  FALSE
P_0004   TRUE  FALSE  FALSE     NA   TRUE   TRUE
P_0005  FALSE   TRUE  FALSE   TRUE     NA  FALSE
P_0006  FALSE  FALSE  FALSE   TRUE  FALSE     NA
> adj.100.200<-adj.100.200*1 #transformando TRUE em 1
> adj.100.200[1:6,1:6]
       P_0001 P_0002 P_0003 P_0004 P_0005 P_0006   ...
P_0001     NA      0      1      1      0      0
P_0002      0     NA      0      0      1      0
P_0003      1      0     NA      0      0      0
P_0004      1      0      0     NA      1      1
P_0005      0      1      0      1     NA      0
P_0006      0      0      0      1      0     NA
```

Encontrando o índice dos pontos mais próximos ordenando as colunas de cada linha, ou seja, usando apply e colocando o índice dos pontos mais próximos primeiros na linha. Ao transpor, a primeira coluna será o próprio ponto e dai por diante podemos selecionar N pontos mais próximos.

Vamos mostra como escolher os três mais próximos:

```
> #três mais próximos
> cols<-apply(distancias,1,order)
> cols<-t(cols)
> cols<-cols[,2:4] # 3 pontos colunas 2,3 e 4
> rowcols<-cbind(rep(1:nrow(geoqui),each=3),as.vector(t(cols)))
> prox3<-adj.100.200*0 #criando matriz de zeros com diagonal NA
> prox3[rowcols]<-1 #assinalando 1 aos índices dos pontos mais próximos
> prox3[1:13,1:10]
       P_0001 P_0002 P_0003 P_0004 P_0005 P_0006 P_0007 P_0008 P_0009 P_0010
P_0001     NA      0      0      0      0      0      0      0      0      0
P_0002      0     NA      0      0      0      0      0      0      0      0
P_0003      0      0     NA      0      0      0      1      0      0      0
P_0004      0      0      0     NA      0      0      0      0      1      0
P_0005      0      0      0      0     NA      0      0      0      0      0
P_0006      0      0      0      0      0     NA      0      0      0      0
P_0007      0      0      1      0      0      0     NA      0      0      0
P_0008      0      0      0      0      0      0      0     NA      0      0
P_0009      0      0      0      1      0      0      0      0     NA      0
P_0010      0      0      0      0      0      0      0      0      0     NA
P_0011      0      0      0      0      0      0      0      0      0      0
P_0012      0      0      0      0      0      0      0      0      0      0
P_0013      0      0      0      1      0      0      0      0      1      0
```

Criando uma matrix do inverso da distância (peso) e normalizando o resultado para 1 (a soma de todas as distâncias com um ponto, expressos pela linha, será 1):

```
> #matrix inverso das distâncias (peso)
> w<-1/distancias
> w[!is.finite(w)] <- NA
> rtot<-apply(w,1,sum,na.rm=TRUE)
> wnorm<-round(w/rtot,8)
> wnorm[1:6,1:6]
           P_0001     P_0002     P_0003     P_0004     P_0005     P_0006
P_0001         NA 0.00078951 0.00188174 0.00112376 0.00061422 0.00065903
P_0002 0.00070992         NA 0.00065113 0.00238500 0.00190301 0.00296400
P_0003 0.00191846 0.00073825         NA 0.00097135 0.00055924 0.00060847
P_0004 0.00102953 0.00242995 0.00087287         NA 0.00118084 0.00143087
P_0005 0.00053249 0.00183475 0.00047555 0.00111742         NA 0.00485839
P_0006 0.00057389 0.00287041 0.00051972 0.00136005 0.00488003         NA
> sum(wnorm[451,],na.rm=TRUE)
[1] 1
> sum(wnorm[3081,],na.rm=TRUE)
[1] 1
```

3.1.3 - Interpolação de dados pontuais

Vamos carregar os dados e criar um objeto SpatialPoints DataFrame com eles:

```
> geoqui<-read.csv('usa_quim_solo.csv')
> library(raster)
> coordinates(geoqui)<-~Longitude+Latitude
```

Carregando dado vector do arquivo us.shp que será a máscara para os dados do objeto geoqui:

```
> library(rgdal)
> us.sp<-readOGR(dsn='.',layer='us')
```

Transformando os dois objetos para um mesmo CRS (WGS-84):

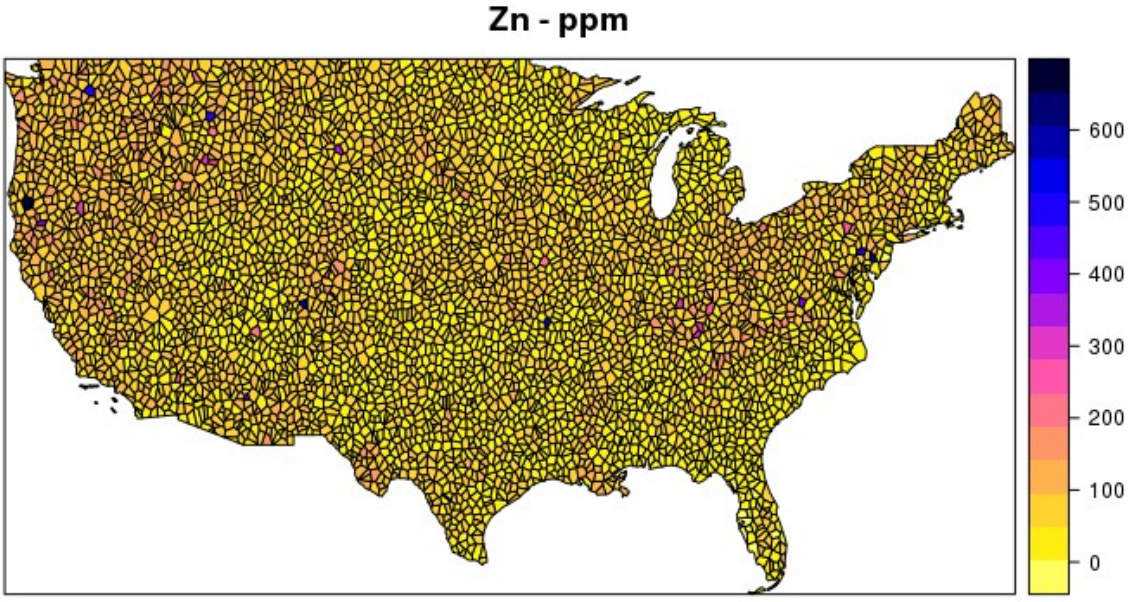
```
> t.str <- CRS('+init=epsg:4326')
> crs(geoqui)<-CRS('+init=epsg:4269')
> geoqui.sp<-spTransform(geoqui, t.str)
> crs(us.sp)
CRS arguments:
 +init=epsg:4326 +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84
+towgs84=0,0,0
> crs(geoqui.sp)
CRS arguments:
 +init=epsg:4326 +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84
+towgs84=0,0,0
```

Plotando o mapa dos dados que usaremos:

```
> plot(us.sp)
> plot(geoqui.sp,add=TRUE, pch=20, col='red',cex=0.1)
```

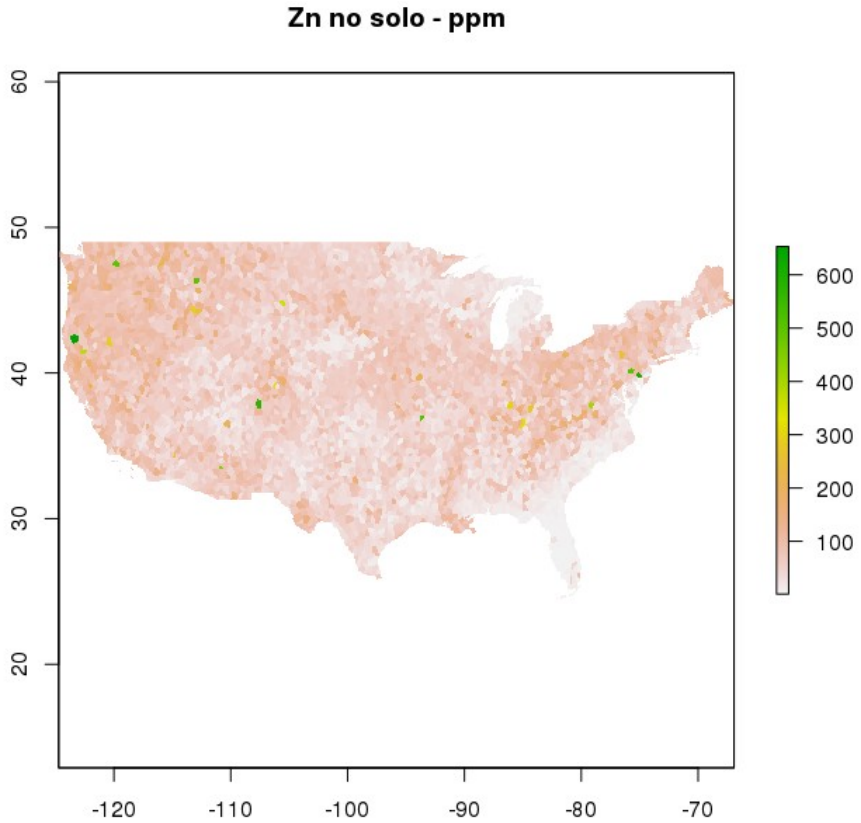
Vamos agora criar polígonos de proximidade usando **voronoi** e usar objeto us.sp como mascara:

```
> library(dismo)
> v<-voronoi(geoqui.sp)
> vor.us<-intersect(v,us.sp)
> spplot(vor.us,'Zn.ppm',col.regions=rev(get_col_regions()), main="Zn - ppm")
```



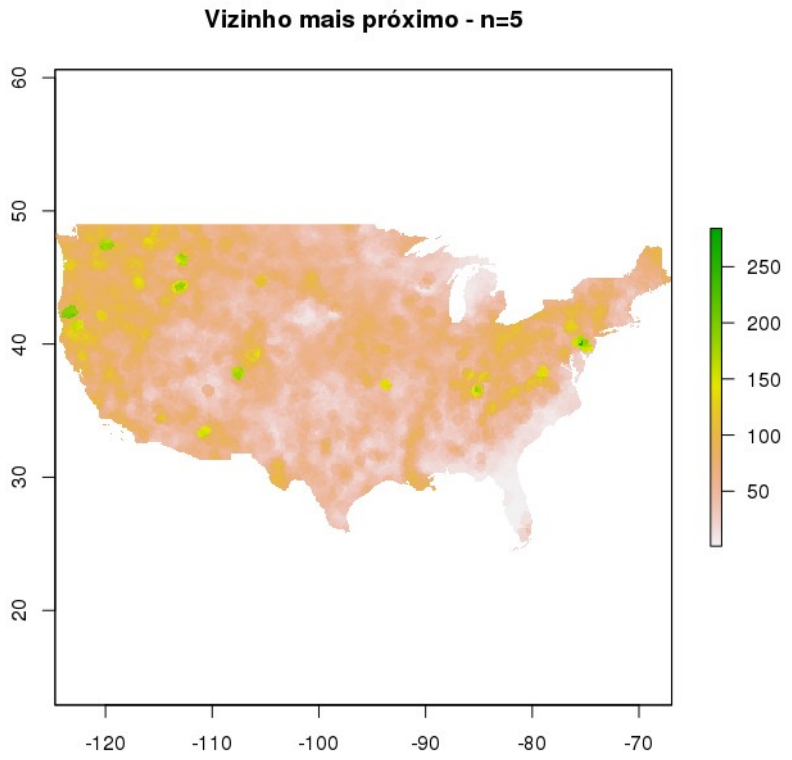
E agora podemos gerar um raster deste polígono usando rasterize:

```
> rast<-raster(us.sp,res=0.05)
> v.rast<-rasterize(vor.us,rast,'Zn.ppm') #demora um pouco….
> plot(v.rast,main='Zn no solo - ppm')
```



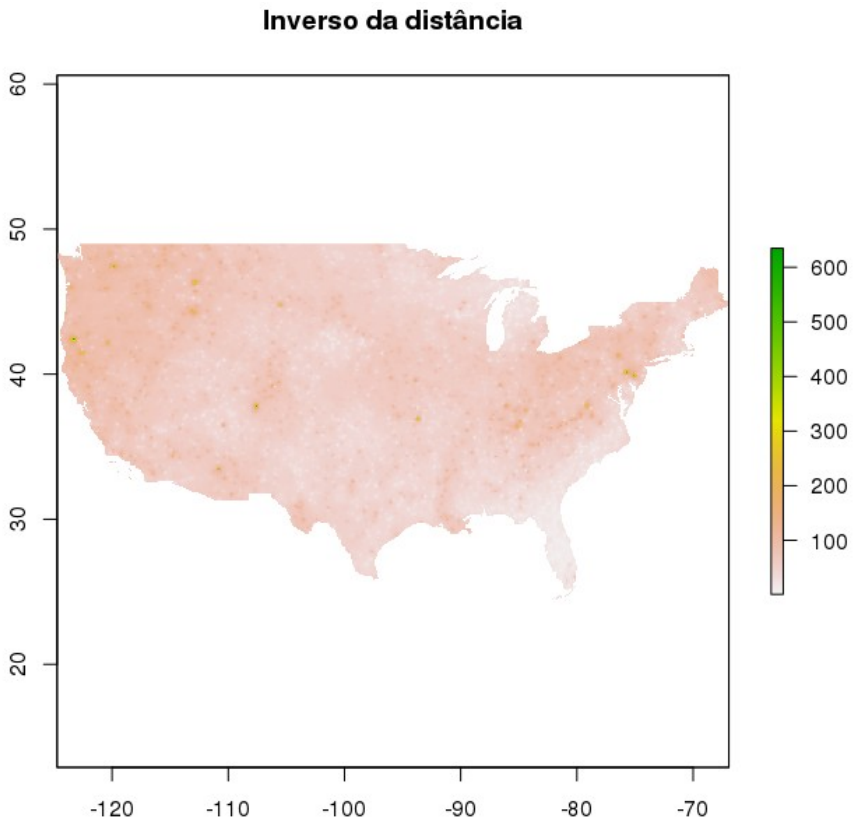
Faremos agora a interpolação usando o método do **vizinho mais próximo** nos dados de Zn.ppm. Usaremos o pacote gstat para isso e utilizaremos um número de 5 vizinhos para o procedimento:

```
> rast<-raster(us.sp,res=0.05)
> rast<-projectRaster(rast,crs='+init=epsg:4326')
> library(gstat)
> metodo<-gstat(formula=Zn.ppm~1,locations=geoqui.sp,nmax=5,set=list(idp=0))
> vmp<-interpolate(rast,metodo) #demora um pouco 15 min
> vmpmsc<-mask(vmp,us.sp)
> plot(vmpmsc,main='Vizinho mais próximo - n=5')
```



Efetuaremos agora a interpolação de Zn.ppm usando o **inverso da distância**. A diferença entre inverso de distância e Vizinho mais próximo é que esta dá menos pesos aos pontos mais distante quando interpolando os valores:

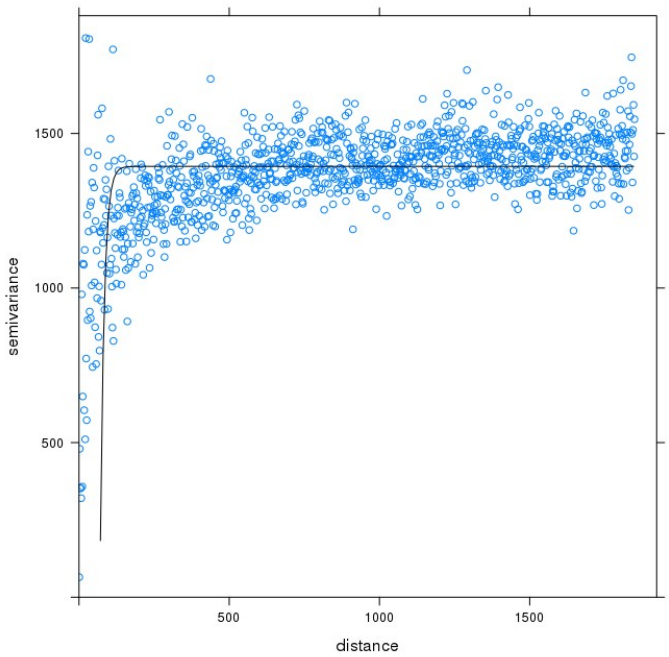
```
> rast<-raster(us.sp,res=0.05)
> rast<-projectRaster(rast,crs='+init=epsg:4326')
> library(gstat)
> metodo <- gstat(formula=Zn.ppm~1, locations=geoqui.sp)
> idw<-interpolate(rast,metodo) # demora um pouco  10 minutos
[inverse distance weighted interpolation]
> idwmsc<-mask(idw,us.sp)
> plot(idwmsc,main='Inverso da distância')
```



Efetuaremos agora a interpolação de Zn.ppm usando o método **krigagem**. Para utilizar krigagem a primeira coisa que precisamos é estabelecer um variograma.

O pacote gstat nos permite efetuar o variograma da seguinte forma e obtemos um modelo que melhor se encaixa ao variograma usando::

```
> krig.met <- gstat(formula=Zn.ppm~1, locations=geoqui.sp)
> vario <- variogram(krig.met, width=1.65)
> f.v <- fit.variogram(vario, vgm(c("Sph","Gau","Exp"))
> plot(variogramLine(f.v, 2000), type='l', ylim=c(0,1600))
> plot(vario)
> lines(variogramLine(f.v, 2000), type='l', ylim=c(0,1600))
```



E agora usando o modelo acima efetuaremos a krigagem ordinária usando:

```
> krig<- gstat(formula=Zn.ppm~1, locations=geoqui.sp, model=f.v)
```

Preparando um grid para interpolar a krigagem

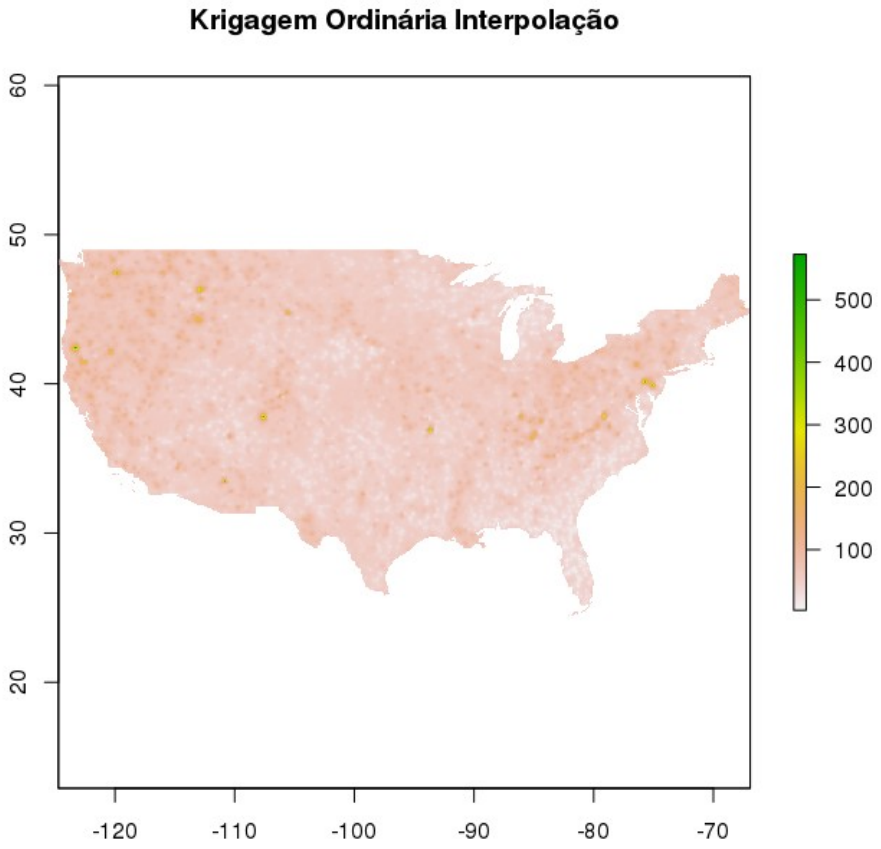
```
> us.r<-raster(us.sp)
> res(us.r)<-0.05
> us.gri <- as(us.r, 'SpatialGrid')
```

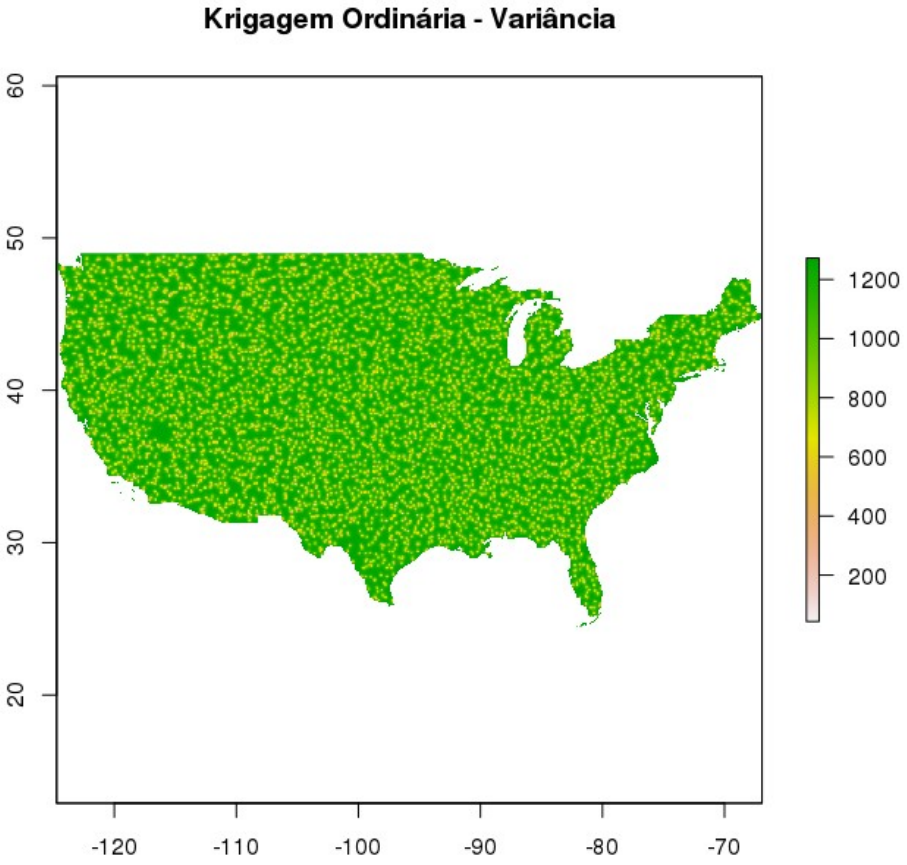
Executando krigagem (usando resolução com valores menores que 0.05 demorará mais de 5 horas):

```
> kri.pred<-predict(krig,us.gri) # vai demorar muito. Quase 7 horas no notebook
[using ordinary kriging]
```

Plotando o resultado de interpolação e da variância

```
> ko <- brick(kri.pred)
> ko <- mask(ko, us.sp)
> names(ko) <- c('Interpolação', 'Variância')
> plot(ko$Interpolação, main='Krigagem Ordinária Interpolação')
> plot(ko$Variância, main= 'Krigagem Ordinária - Variância')
```




Krigagem Ordinária - Variância
60
50
40
30
20
-120
-110
-100
-90
-80
-70
1200
1000
800
600
400
200

3.1.4 - Estatística Prática com Dados Pontuais - Distribuição dos dados

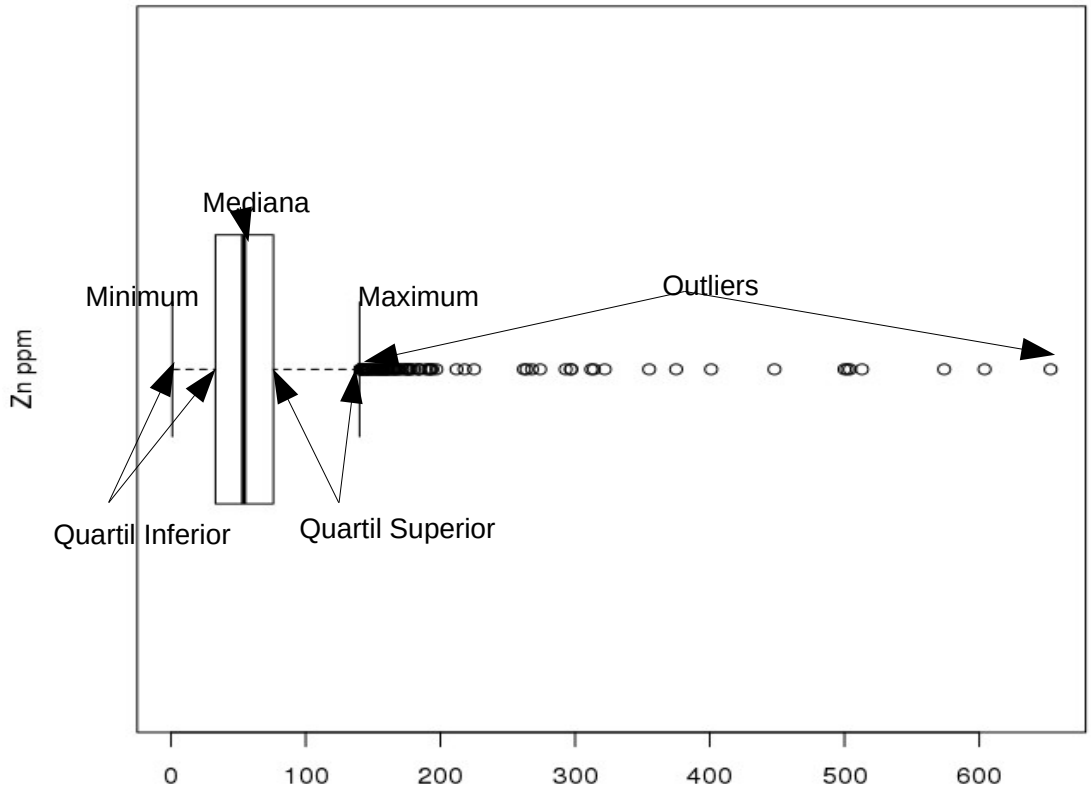
Cobriremos aqui os aspectos essenciais de estatística para análise espacial de dados geológicos pontuais. Algumas dicas de como avaliar os dados estatísticos serão apresentadas também.

Primeiramente temos que conhecer os dados que trabalharemos para evitar situações como tivemos acima (distribuição Normal x Poisson) que podem levar a erros futuros caso tomemos interpretações erradas no início.

Uma maneira eficiente para resolver isso é usando o pacote vioplot que é um pacote que faz o mesmo que o boxplot mas com uma informação a mais.

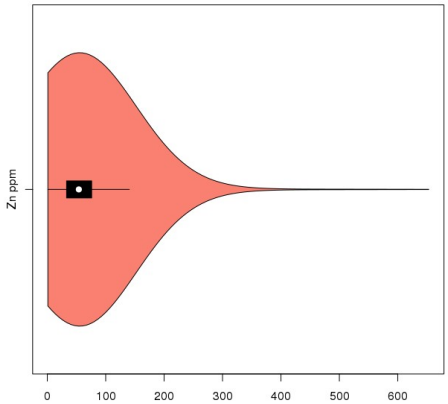
Antes vejamos o que o boxplot fornece:

```
> geoqui<-read.csv('usa_quim_solo.csv')
> boxplot(geoqui[47],horizontal=TRUE,ylab='Zn ppm')
```



Usando o vioplot temos uma melhor representação da distribuição deste dados:

```
> library(vioplot)
> vioplot(geoqui$Zn.ppm,names=c("Zn ppm"),col = "salmon",horizontal=TRUE)
```

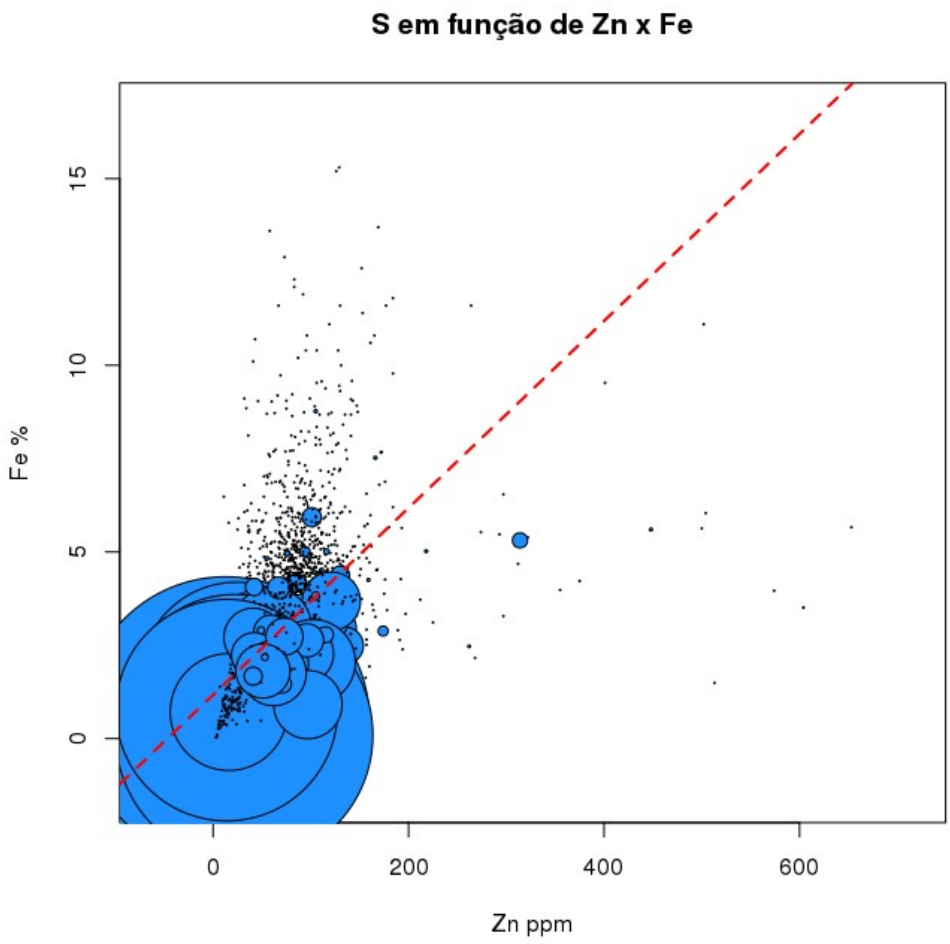


O gráfico acima nos dá de forma prática a curva de distribuição, nesse caso uma distribuição Poisson de fato.

Outra representação gráfica bastante útil em análise espacial é usando duas variáveis e vendo suas correlações através da linha de regressão. Podemos até mesmo usar uma terceira variável para expressar “tridimensionalmente”.

Vamos criar um gráfico de teores de zinco e de ferro das amostras e também usaremos os teores de enxofre expressos no tamanho dos símbolos plotados. Adicionaremos também a reta de regressão usando a função lm.

```
> symbols(geoqui$Zn.ppm,geoqui$Fe.pct,circles=geoqui$S.pct, fg='black',
bg='dodgerblue',xlab='Zn ppm',ylab='Fe %', main ='S em função de Zn x Fe')
> abline(lm(geoqui$Fe.pct~geoqui$Zn.ppm), col="red", lwd=2, lty=2)
> lm(geoqui$Fe.pct~geoqui$Zn.ppm)

Call:
lm(formula = geoqui$Fe.pct ~ geoqui$Zn.ppm)

Coefficients:
  (Intercept)  geoqui$Zn.ppm
      1.17006        0.02505
```



3.1.5 - Estatística Prática com Dados Pontuais – Regressão Linear

Vamos dar prosseguimento inspecionando como avaliar o relacionamento entre duas ou mais variáveis de um conjunto de dados.

Para isso vamos usar dois elementos acima aparentemente correlacionados do poto de vista geoquímico, zinco e ferro.

Um coeficiente de correlação mede o grau pelo qual duas variáveis tendem a mudar juntas. Coeficiente próximo a 1 representa uma correlação positiva, próximo a zero não estão correlacionados e próximo a -1 estão inversamente correlacionados. A correlação de Pearson avalia a relação linear entre duas variáveis contínuas. Uma relação é linear quando a mudança em uma variável é associada a uma mudança proporcional na outra variável. A correlação de Spearman avalia a relação monotônica entre duas variáveis contínuas ou ordinais. Em uma relação monotônica, as variáveis tendem a mudar juntas mas não necessariamente a uma taxa constante. O coeficiente de correlação de Spearman baseia-se nos valores classificados de cada variável, ao invés dos dados brutos.

Vejamos as correlações usando cor:

```
> cor(geoqui$Zn.ppm,geoqui$Fe.pct,method='pearson')
[1] 0.6008144
> cor(geoqui$Zn.ppm,geoqui$Fe.pct,method='spearman')
[1] 0.7883986
```

Ou detalhadamente usando cor.test:

```
> cor.test(geoqui$Zn.ppm,geoqui$Fe.pct,method='pearson')
        Pearson's product-moment correlation
data:  geoqui$Zn.ppm and geoqui$Fe.pct
t = 51.952, df = 4778, p-value < 2.2e-16
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.5823842 0.6186273
sample estimates:
      cor
0.6008144
> cor.test(geoqui$Zn.ppm,geoqui$Fe.pct,method='spearman')
        Spearman's rank correlation rho
data:  geoqui$Zn.ppm and geoqui$Fe.pct
S = 3851700000, p-value < 2.2e-16
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.7883986

Warning message:
In cor.test.default(geoqui$Zn.ppm, geoqui$Fe.pct, method = "spearman") :
  Impossível calcular o valor exato de p com empates
```

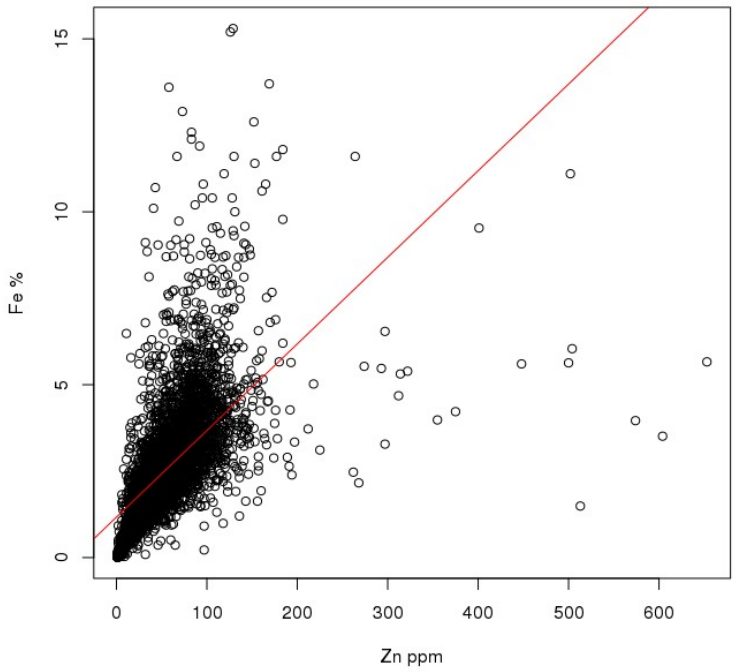
Podemos criar uma matriz de correlações para todos os elementos dos dados e selecionar a correlação de um elemento com cada elemento em ordem decrescente usando:

```
> tudo<-geoqui[,4:47]
> correlacoes<-cor(tudo,method='pearson')
> rev(sort(correlacoes[44,])) #44 é zinco
     Zn.ppm      Fe.pct      Ga.ppm      Al.pct      Sc.ppm       V.ppm
 1.00000000  0.60081441  0.55605473  0.55520452  0.54741777  0.52680784
     Sn.ppm      Ti.pct      Co.ppm       Y.ppm      Ti.ppm      Mn.ppm
```

```
 0.49705005  0.45239060  0.44675417  0.43804768  0.42821984  0.42761526
     Li.ppm      Be.ppm      Pb.ppm      In.ppm      Cu.ppm      Rb.ppm
 0.42480162  0.41138166  0.39357017  0.38659576  0.37059981  0.36059223
     Ce.ppm      La.ppm       P.ppm      Nb.ppm      As.ppm      Sb.ppm
 0.35381672  0.34833541  0.33687067  0.32909896  0.30500886  0.28973070
     Bi.ppm       K.pct      Th.ppm      Hg.ppm      Ba.ppm       U.ppm
 0.28445789  0.28402580  0.26310307  0.25606535  0.25569532  0.24743384
     Mo.ppm      Mg.pct      Na.pct      Cd.ppm      Se.ppm       W.ppm
 0.24300343  0.22446350  0.21838424  0.18328968  0.18224070  0.16932585
     Cr.ppm      Ni.ppm      Te.ppm      Cs.ppm      Ag.ppm      Sr.ppm
 0.16286328  0.16148903  0.15792206  0.13576632  0.09281092  0.03969483
      S.pct       Ca.pct
-0.03010370 -0.06151240
> rev(sort(correlacoes[14,])) #14 é ferro
      Fe.pct       Sc.ppm        V.ppm       Ti.pct       Al.pct       Ga.ppm
 1.000000000  0.873381355  0.793562343  0.783294588  0.734844430  0.707517622
      Co.ppm       Zn.ppm       Mn.ppm        Y.ppm       Sn.ppm       Ce.ppm
 0.683944062  0.600814415  0.423387630  0.421944294  0.418331778  0.364104479
      Be.ppm       Nb.ppm       Cr.ppm        P.ppm       In.ppm       La.ppm
 0.357901184  0.355508822  0.331329209  0.331066244  0.324936202  0.317465279
      Li.ppm       Ni.ppm       Cu.ppm       Mg.pct       Th.ppm       As.ppm
 0.316543836  0.314116171  0.303374608  0.267238422  0.258718574  0.244323905
      Bi.ppm       Ti.ppm       Se.ppm       Na.pct       Hg.ppm        U.ppm
 0.221409059  0.219674947  0.196615810  0.193365112  0.187078660  0.181650445
      Rb.ppm       Mo.ppm       Ba.ppm       Pb.ppm       Sb.ppm        W.ppm
 0.172316797  0.165465434  0.156276753  0.143219516  0.133278298  0.105788740
       K.pct       Cs.ppm       Te.ppm       Ag.ppm       Sr.ppm       Cd.ppm
 0.095176686  0.089860430  0.057751082  0.024612855  0.009625497  0.005511356
       S.pct       Ca.pct
-0.069602690 -0.141559461
```

Vamos agora explorar mais os modelos de regressão linear criados pela função lm. Vamos usar Fe e zinco inicialmente:

```
> modelo1<-lm(geoqui$Fe.pct~geoqui$Zn.ppm)
> plot(geoqui$Zn.ppm,geoqui$Fe.pct,xlab='Zn ppm',ylab='Fe %')
> abline(modelo1,col='red')
```



Visualizando as informações do nosso modelo de regressão linear:

```
> summary(modelo1)
Call:
lm(formula = geoqui$Fe.pct ~ geoqui$Zn.ppm)
Residuals:
     Min       1Q   Median       3Q      Max
-12.7888  -0.6828  -0.2951   0.3347  10.9772
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   1.1700596  0.0339982   34.41   <2e-16 ***
geoqui$Zn.ppm 0.0250476  0.0004821   51.95   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 1.333 on 4778 degrees of freedom
Multiple R-squared:  0.361,     Adjusted R-squared:  0.3608
F-statistic:  2699 on 1 and 4778 DF,  p-value: < 2.2e-16
```

Interpretando o resultado apresentado pela função summary temos:
Call que é a fórmula de chamada do modelo linear onde ***geoqui$Fe.pct*** é a variável dependente (alvo) e ***geoqui$Zn.ppm*** é a variável independente (predictor).
Residuals representam essencialmente a diferença entre o valor observado e o valor indicado pelo modelo linear divididos em cinco pontos (mínimo, primeiro quartil, mediana, terceiro quartil e máximo). Quanto mais simétricos forem estes valores mais correlacionáveis são os valores avaliados. Idealmente mediana deveria ser 0 e os valores mínimo e 1Q iguais com sinais opostos ao máximo e 3Q.
Coefficients são os coeficientes da regressão linear entre os valores avaliados. Na coluna estimate temos o intercepto que é o valor de Fe quando Zn é igual a 0 e o valor de ferro aumenta em 0.0250476 a cada aumento de 1 no teor de zinco. Ou seja, ***Fe=1.1700596 + Zn*0.0250476***. Std error é o erro de cada um destes parâmetros. O coeficiente Valor t expressa quantos desvios padrão esse coeficiente estima estar distante de zero, quanto mais distante (quanto maior o valor) mais correlacionáveis os valores avaliados serão. Pr(>|t|) representa a probabilidade de encontrarmos um valor maior ou igual a t, quanto menor esse valor, mais correlacionáveis os valores serão (p-value<0.05).
Residual standard error é a medida da qualidade de ajuste da qualidade da regressão linear. O erro padrão residual é a media na qual a resposta (Fe) vai desviar do valor da reta de regressão verdadeira.
Multiple R squared é a medida estatística do quanto próximo os dados estão de reta de regressão ajustada. É também conhecida como o coeficiente de determinação. É a medida da relação linear entre nosso indicador (Fe) e nossa variável alvo (Zn). Este valor é sempre entre 0 e 1. No nosso caso estamos 36% próximos.
F-statistic é um bom indicador de que existe uma relação entre nossas variáveis indicador e alvo. O mais distante F-statistic é de 1 melhor é.

Na regressão múltipla podemos usar mais de uma variável independente (predictors) para construir modelos melhores. No nosso caso podemos adicionar uma outra variável para criar um modelo de avaliação.

Escolhemos escândio como segundo predictor uma vez que este possui boa correlação individuas tanto com ferro como com zinco:

```
> cor(geoqui$Fe.pct,geoqui$Zn.ppm + geoqui$Sc.ppm) #pearson é default
[1] 0.6692415
> cor(geoqui$Fe.pct,geoqui$Zn.ppm + geoqui$Sc.ppm,method='spearman')
[1] 0.8290926
```

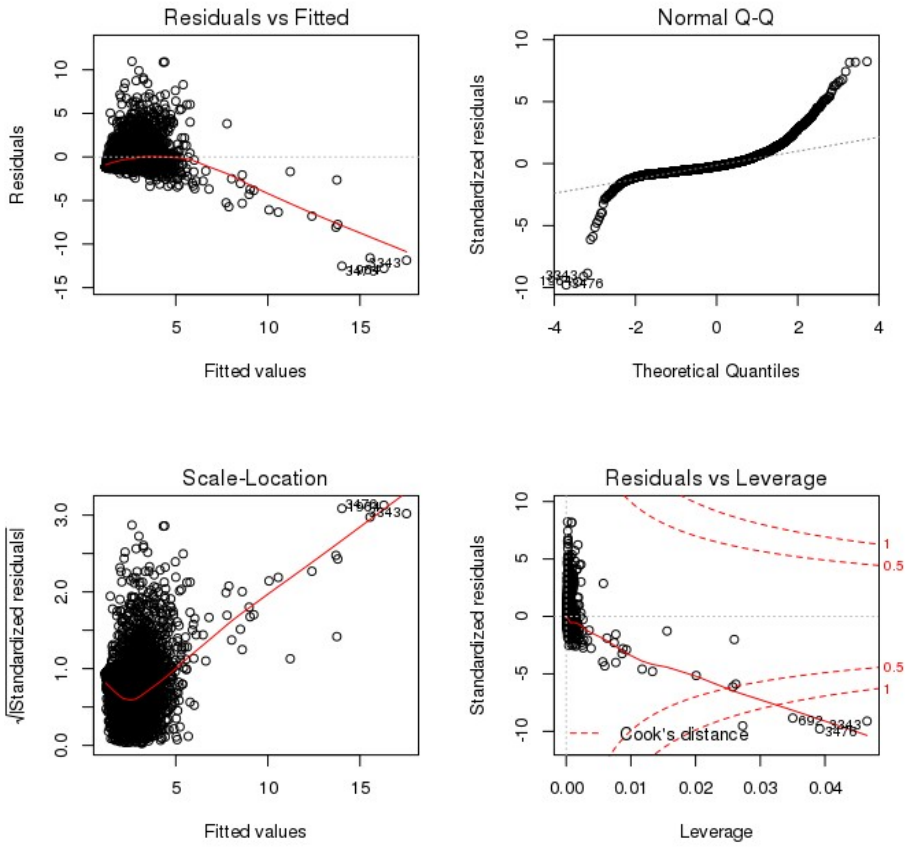
Vendo o sumário do modelo:

```
> modelo2<-lm(geoqui$Fe.pct~geoqui$Zn.ppm + geoqui$Sc.ppm)
> summary(modelo2)

Call:
lm(formula = geoqui$Fe.pct ~ geoqui$Zn.ppm + geoqui$Sc.ppm)
Residuals:
    Min      1Q  Median      3Q     Max
-6.6575 -0.3057 -0.1077  0.1729 10.6261
Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.3173714  0.0216289   14.67   <2e-16 ***
geoqui$Zn.ppm 0.0073047  0.0003348   21.82   <2e-16 ***
geoqui$Sc.ppm 0.2254145  0.0023281   96.82   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 0.7746 on 4777 degrees of freedom
Multiple R-squared:  0.7843,    Adjusted R-squared:  0.7842
F-statistic:  8685 on 2 and 4777 DF,  p-value: < 2.2e-16
```

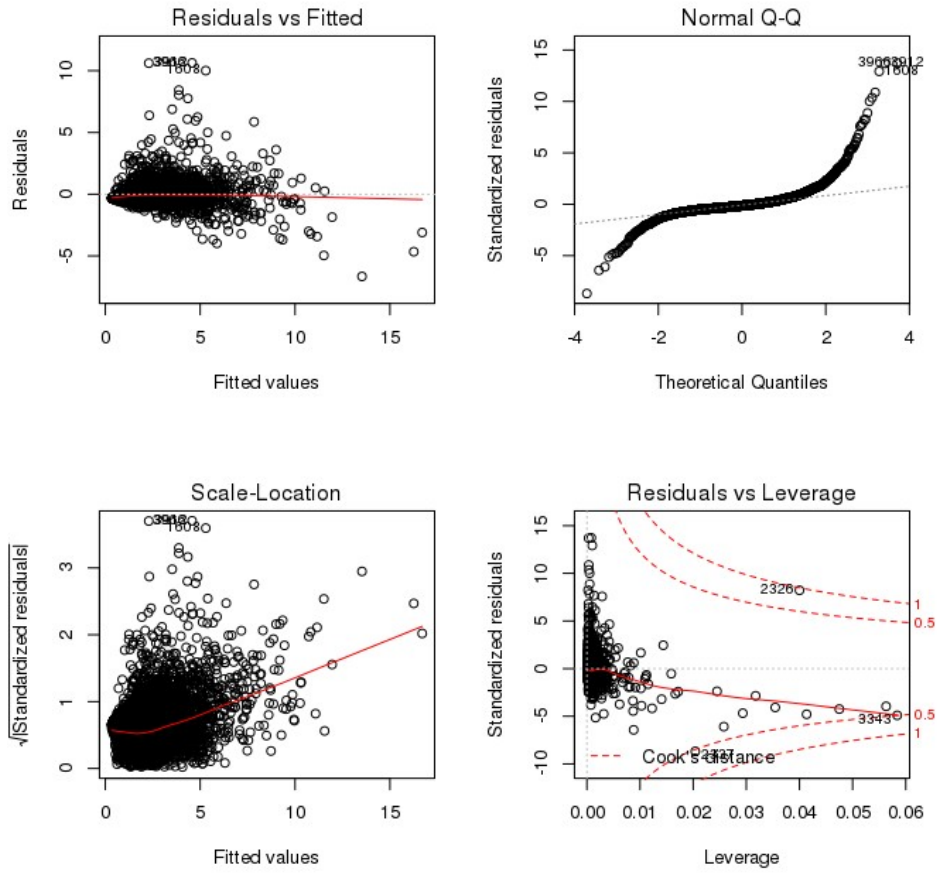
Vamos agora ver como interpretar os gráficos de um modelo linear usando plot com o modelo criado como argumento. Serão gerados 4 gráficos (Residuals x Fitted, Normal Q-Q, Scale x Location e Residuals x Leverage). Usaremos os dois modelos criados para ver como ele melhorou (ou não).

Plotando os modelos:

```
> par(mfrow=c(2,2))
> plot(modelo1)
```



```
> plot(modelo2)
```



O gráfico residuals x fitted tem padrão não linear. Se ele apresentar residuais espalhados em volta de uma linha horizontal com padrões distintos é uma boa indicação de que você tem uma relação linear entre preditor e variável predita. Nosso segundo modelo (regressão múltipla) apresenta um melhor resultado com relação ao primeiro modelo (regressão simples).

O segundo gráfico mostra se os dados residuais estão distribuídos de forma normal. Quanto mais dados estiverem alinhados com a linha pontilhada melhor é o modelo de regressão. Podemos ver que o segundo modelo é ligeiramente melhor nesse aspecto.

O gráfico scale x location também chamado de gráfico de espalhamento mostra se os residuais estão distribuídos ao longo das faixas de valores dos preditores. Quanto mais espalhado ao longo de uma reta mais horizontal melhor é a regressão. Novamente modelo dois é melhor um pouco.

O último gráfico (residual x leverage) nos ajuda a encontrar casos alterantes do modelo. Nem todo valor extremo (outlier) é alterante do modelo, ou seja, ele não alteraria à regressão se fossem excluídos. Já alguns valores extremos alterarão mais intensamente o resultado da regressão se fossem excluídos (são off trend). O padrão de distribuição dos pontos nesse gráfico é irrelevante, ao contrário dos gráficos anteriores. Aqui checamos se existem valores que vão cair no canto superior e inferior direito fora do limite da distância de Cook e portanto fortemente alterando o resultado da regressão. Nosso segundo modelo reduziu bastante está influência.
Chegamos à conclusão de que o segundo modelo usando Zn e Sc como preditores geraram uma melhor regressão para o valor de Fe do que usando somente Zn como preditor.

Vamos agora testar este modelo usando um outro conjunto de dados. Vamos testar ele contra o resultado das análises do horizonte A efetuado pelo USGS. Vamos calcular os teores de Fe usando nossa curva de regressão do modelo com os valores Zn e Sc e depois comparar com os valores reais de Fe do Horizonte A.

Carregando dados e criando Coeficientes do modelo:

```
> geoqui<-read.csv('usa_quim_solo.csv')
> library(raster)
> coordinates(geoqui)<-~Longitude+Latitude
> library(rgdal)
> us.sp<-readOGR(dsn='.',layer='us')
> t.str <- CRS('+init=epsg:4326')
> crs(geoqui)<-CRS('+init=epsg:4269')
> geoqui.sp<-spTransform(geoqui, t.str)
> modelo2<-lm(geoqui$Fe.pct~geoqui$Zn.ppm + geoqui$Sc.ppm)
> modelo2$coefficients
     (Intercept) geoqui.sp$Zn.ppm geoqui.sp$Sc.ppm
     0.317371447      0.007304679      0.225414465
```

Nossa equação é (do modelo dos dados do horizonte C acima):

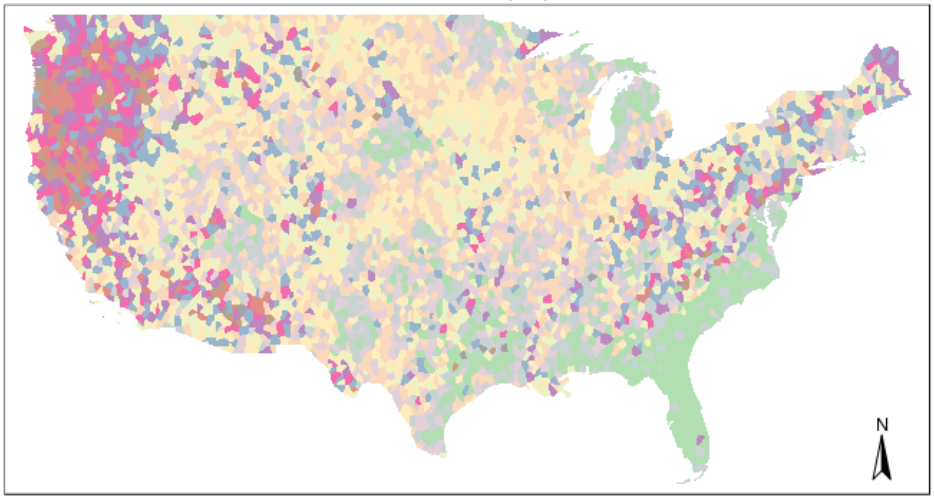
Fe = 0,31 + Zn*0.007304679 + Sc*0.225414465

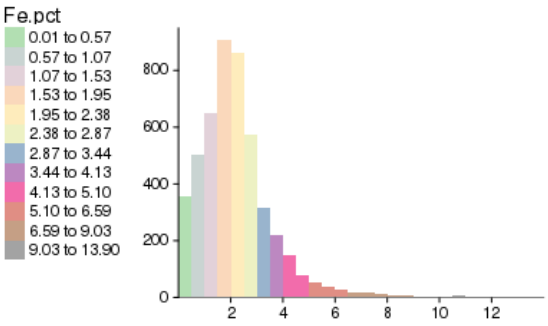
```
> HzA<-read.csv('Ahorizon.csv')
> library(raster)
> coordinates(HzA)<-~Longitude+Latitude
> crs(HzA)<-CRS('+init=epsg:4269')
> HzA.sp<-spTransform(HzA, t.str)
> head(HzA.sp,n=1)
  Fe.pct Sc.ppm Zn.ppm
1   0.57    1.6     15
> HzA.sp$Fe.pred<-modelo2$coefficients[1]+modelo2$coefficients[2]*HzA.sp$Zn.ppm
+modelo2$coefficients[3]*HzA.sp$Sc.ppm
> head(HzA.sp,n=1)
  Fe.pct Sc.ppm Zn.ppm   Fe.pred
1   0.57    1.6     15 0.7876048
```

Plotando o valor real das análise e o valor predito pelo modelo:

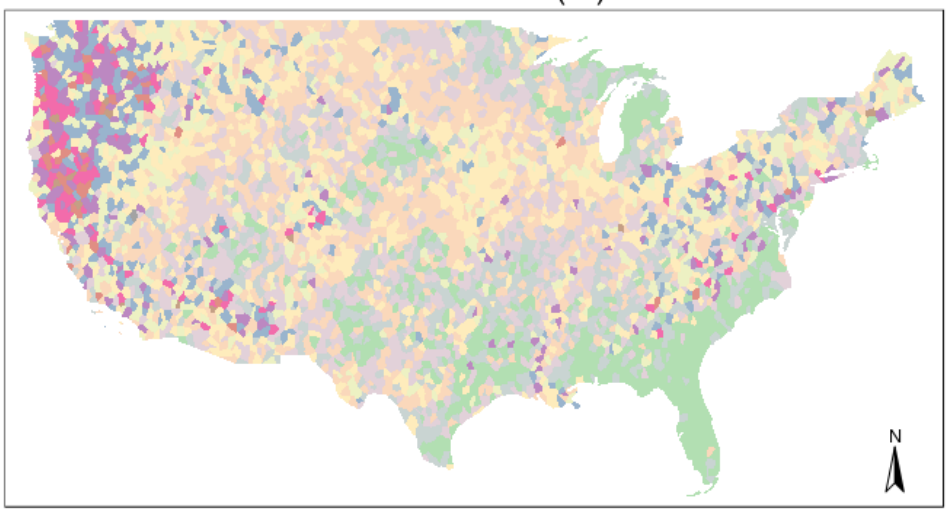
```
> library(dismo)
> v<-voronoi(HzA.sp)
> vor.us<-intersect(v,us.sp)
> library(tmap)
> library(RColorBrewer)
> paleta <- brewer.pal(n=8,name = "Accent")
> tm_shape(vor.us) + tm_fill(alpha=.6,n=12,legend.hist=TRUE,col =
"Fe.pct",style="jenks",palette=paleta)+tm_layout(main.title='Teor Ferro Real do
Horizonte A (%)',legend.outside.position = c("bottom"),legend.outside=
TRUE,legend.stack= 'horizontal', legend.hist.height=1.05, legend.hist.width
=1.05) + tm_compass()
> tm_shape(vor.us) + tm_fill(alpha=.6,n=12,legend.hist=TRUE,col =
"Fe.pred",style="jenks",palette=paleta)+tm_layout(main.title='Teor Ferro
Predito usando modelo (%)', legend.outside.position =
c("bottom"),legend.outside=TRUE, legend.stack= 'horizontal',
legend.hist.height=1.05,legend.hist.width=1.05)+tm_compass()
```

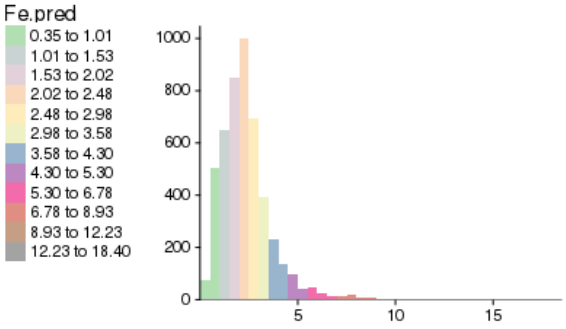
Teor Ferro Real do Horizonte A (%)





Teor Ferro Predito usando modelo (%)





O resultado obtido foi bastante satisfatório pois os dois mapas gerados (real e predito) são bastante similares. Nesse exemplo nós usamos uma regressão múltipla de um conjunto de dados e o aplicamos em um outro conjunto de dados e comprovamos que a regressão é válida e consideravelmente precisa.

3.1.6 - Estatística Prática com Dados Pontuais – Autocorrelação espacial

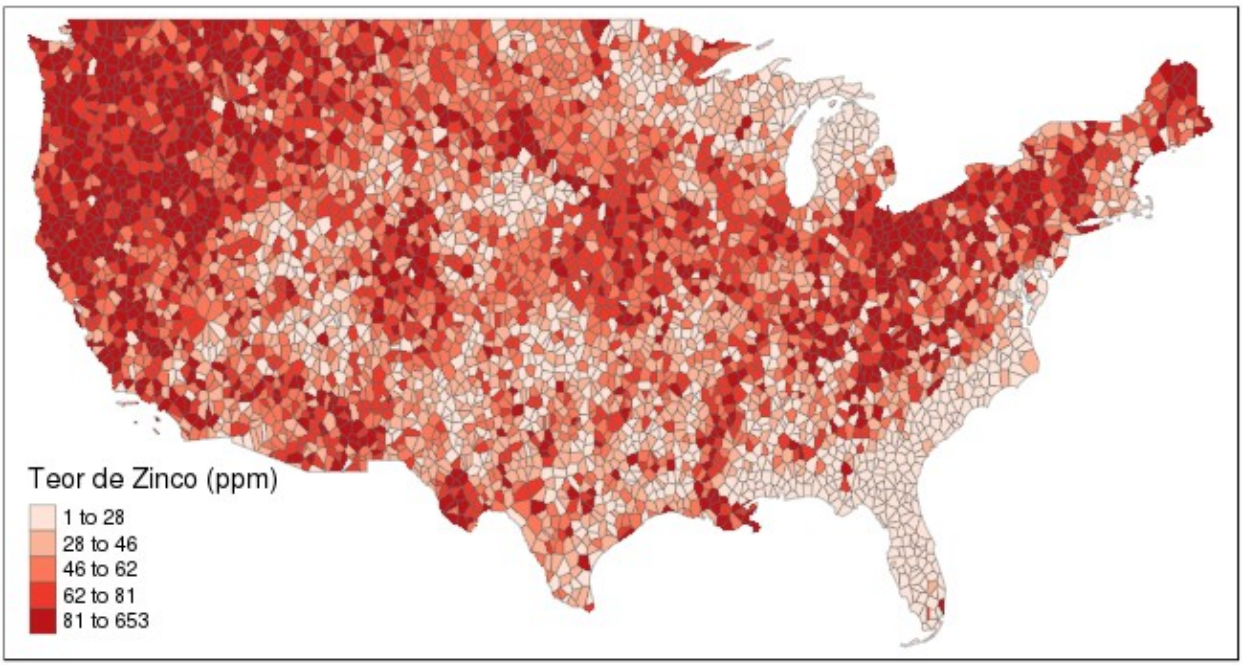
Vamos abordar agora como efetuar medidas da autocorrelação com R. Vamos considerar estatística geral de autocorrelação espacial e como identificar aglomerações ao logo da área amostrada.

Carregando os dados:

```
> geoqui<-read.csv('usa_quim_solo.csv')
> library(raster)
> coordinates(geoqui)<-~Longitude+Latitude
> library(rgdal)
> us.sp<-readOGR(dsn='.',layer='us')
> t.str <- CRS('+init=epsg:4326')
> crs(geoqui)<-CRS('+init=epsg:4269')
> geoqui.sp<-spTransform(geoqui, t.str)
> library(dismo)
> v<-voronoi(geoqui.sp)
> vor.us<-intersect(v,us.sp)
```

Plotando o nosso dado mostrando o teor de Zinco:

```
> library(tmap)
> tm_shape(vor.us) + tm_fill('Zn.ppm', palette = 'Reds', style = 'quantile',
title = 'Teor de Zinco (ppm)') + tm_borders(alpha=.4)
```



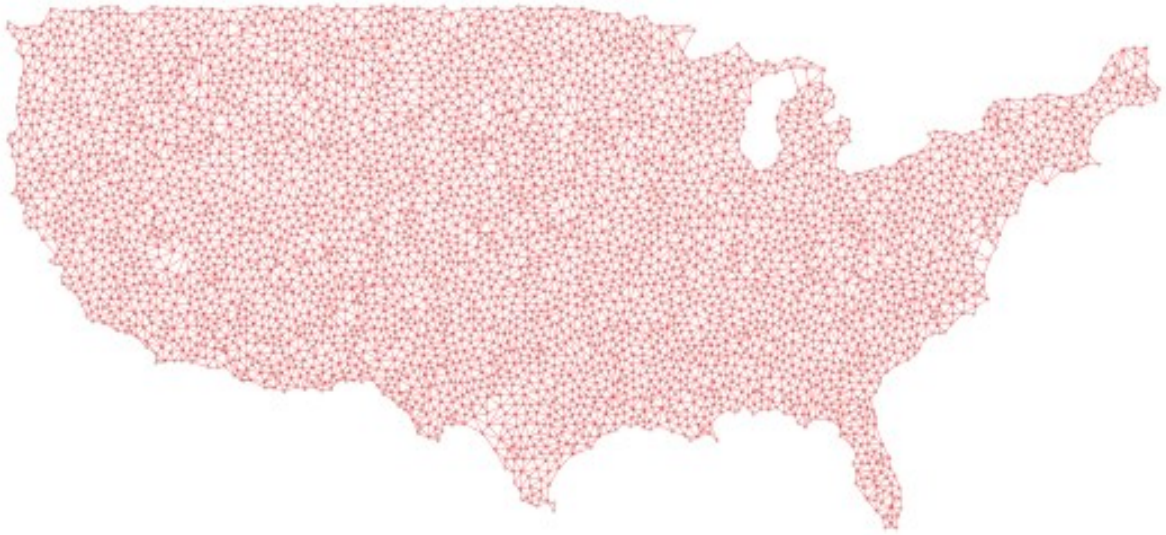
Vamos agora executar a autocorrelação espacial que informa como a distância influencia uma variável em particular e como estes são similares a objetos próximos. As variáveis são ditas terem uma autocorrelação espacial positiva quando valores similares tendem a estar mais próximos espacialmente do que os valores não similares.

Usaremos o pacote spdep para efetuarmos a autocorrelação mas vamos primeiro calcular os vizinhos de nossos dados:

```
> library(spdep)
> vizin<-poly2nb(vor.us)
> vizin
Neighbour list object:
Number of regions: 4780
Number of nonzero links: 27924
Percentage nonzero weights: 0.1222142
Average number of links: 5.841841
```

Plotando eles:

```
> plot(vizin, coordinates(vor.us), col='red',cex=0.2,lwd=0.1)
```



Agora vamos transformar nosso dado de vizinhos para uma lista

```
> lista.vz<-nb2listw(vizin)
> lista.vz
Characteristics of weights list object:
Neighbour list object:
Number of regions: 4780
Number of nonzero links: 27924
Percentage nonzero weights: 0.1222142
Average number of links: 5.841841

Weights style: W
Weights constants summary:
     n       nn   S0       S1       S2
W 4780 22848400 4780 1679.798 19423.64
```

Podemos agora executar nosso modelo. Este modelo é conhecido como **teste de Moran**. Ele criará uma graduação de correlação entre -1 e 1. Tal como um coeficiente de correlação, 1 determina autocorrelação espacial positiva perfeita, 0 indica que os dados são aleatoriamente distribuídos e -1 representa autocorrelação espacial negativa (valores não similares estão próximos uns dos outros).

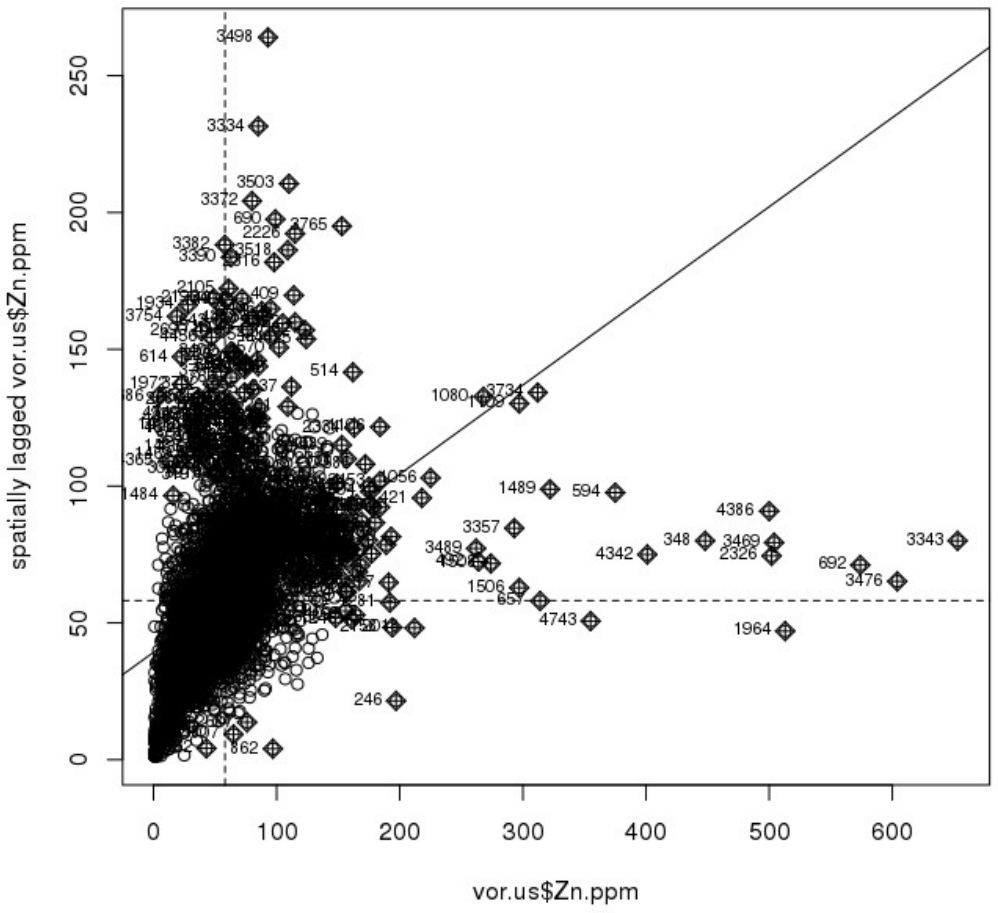
Vamos executar o teste de Moran:

```
> mo<-moran.test(vor.us$Zn.ppm,lista.vz)
> mo
        Moran I test under randomisation

data:  vor.us$Zn.ppm
weights: lista.vz

Moran I statistic standard deviate = 38.203, p-value < 2.2e-16
alternative hypothesis: greater
sample estimates:
Moran I statistic       Expectation          Variance
     3.257159e-01     -2.092488e-04      7.278398e-05
```

A estatística Moran I é 0.325, e podemos dizer que essa variável é ligeiramente autocorrelata com uma leve tendência de gerar aglomerações de dados similares.

Vamos gerar um gráfico que mostra os valores plotados contra sua defasagem espacial na forma de um gráfico de pontos com estilo W padronizados por linha (soma / numero total de ligações):

```
> moran <- moran.plot(vor.us$Zn.ppm, listw = nb2listw(vizin, style = "W"))
```



Agora com a função localmoran calcularemos as estatísticas Moran para cada ponto de teor de zinco:

```
> mo.local <- localmoran(x=vor.us$Zn.ppm, listw = nb2listw(vizin, style = "W"))
> head(mo.local)
         Ii          E.Ii    Var.Ii       Z.Ii  Pr(z > 0)
1  0.6533400 -0.0002092488 0.1649630  1.6091074 0.05379643
2  0.4809709 -0.0002092488 0.2475472  0.9671161 0.16674299
3  0.7667560 -0.0002092488 0.2475472  1.5415110 0.06159623
4  0.3400883 -0.0002092488 0.2475472  0.6839586 0.24700066
5 -0.1928539 -0.0002092488 0.1649630 -0.4743114 0.68236107
6  0.2326794 -0.0002092488 0.1649630  0.5733965 0.28318812
```

Onde:

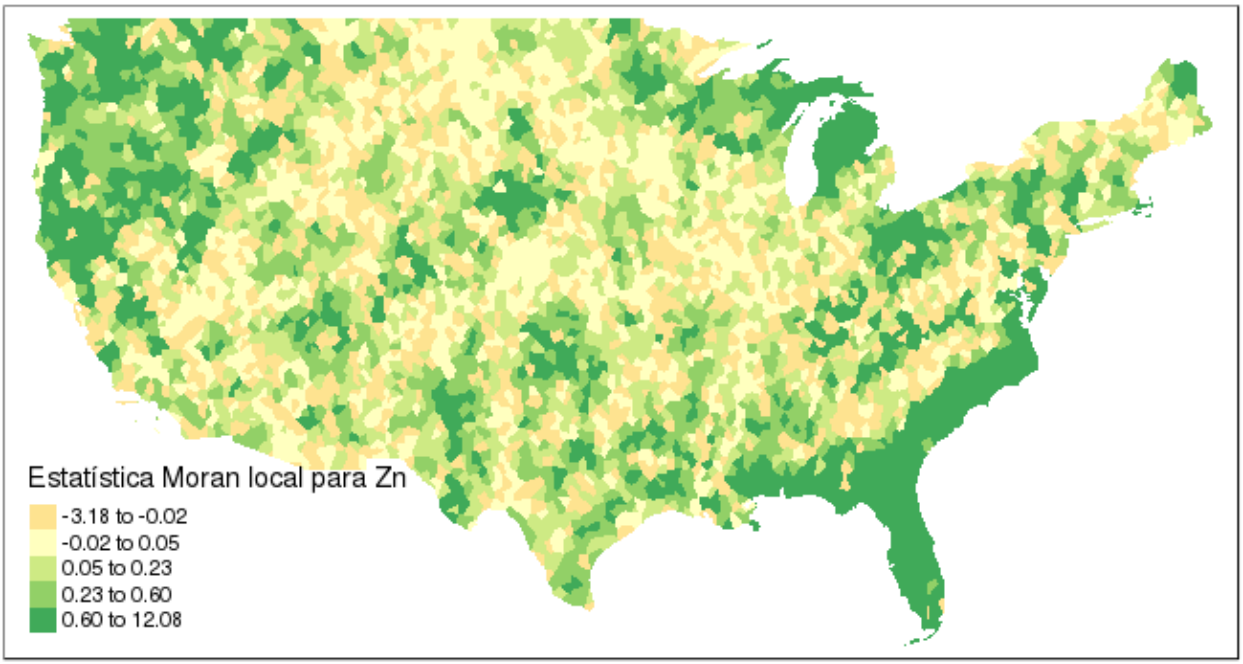
Ii	Estatística Moran do ponto
E.li	Expectativa da estatística Moran local
Var.li	Variância da estatística Moran local
Z.li	Desvio padrão da estatística Moran local
Pr()	p-value de estatística Moran local

Vamos agora adicionar estes parâmetros aos nossos dados

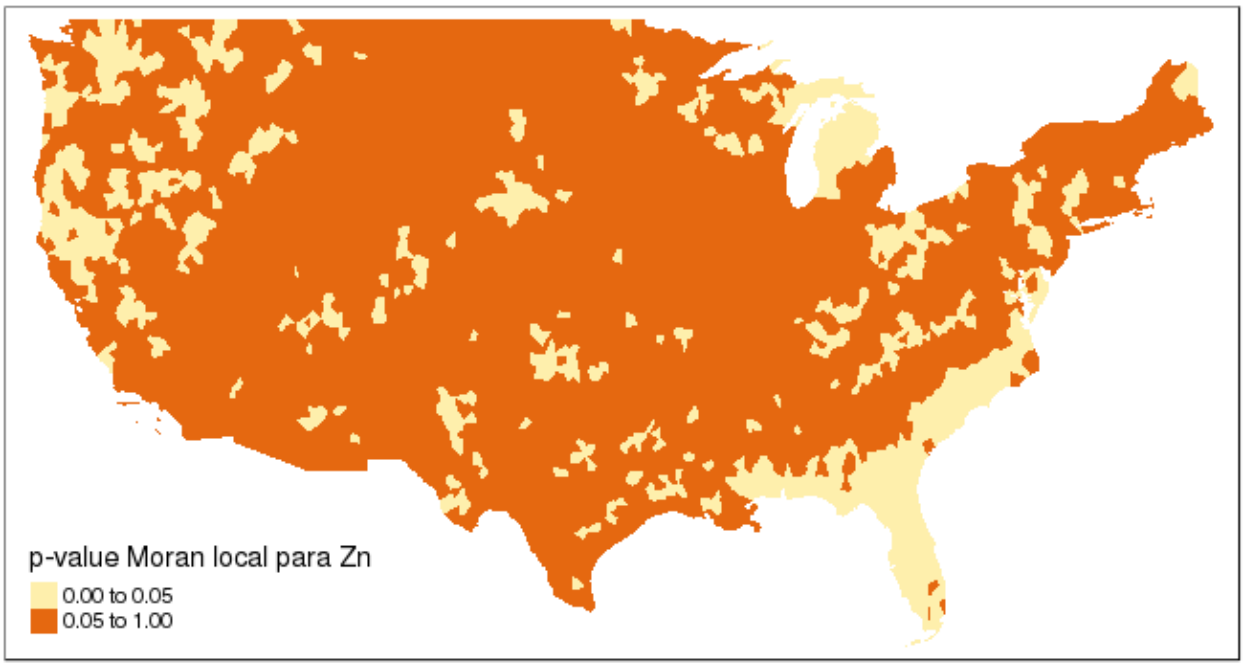
```
> mapa.moran<-cbind(vor.us,mo.local)
```

E plotar alguns deles (Ii e p-value):

```
> tm_shape(mapa.moran) + tm_fill('Ii', style = 'quantile',title = 'Estatística
Moran local')
```



```
> tm_shape(mapa.moran) + tm_fill('Pr.z...0.', n=10, style = 'fixed',breaks=
c(0,0.05,1), title = 'p-value Moran local para Zn')
```



Vamos agora efetuar um mapa que categoriza o tipo de relação cada elemento possui com seu vizinho ou **LISA** (Local Indicators of Spatial Association):

```
> qua<-vector(mode='numeric',length=nrow(mo.local))
> media.Zn<-vor.us$Zn.ppm-mean(vor.us$Zn.ppm)
> media.mo<-mo.local[,1]-mean(mo.local[,1])
> significancia<-0.1
> qua[media.Zn >0 & media.mo>0]<-2
> qua[media.Zn <0 & media.mo>0]<-1
> qua[mo.local[,5] >significancia]<-0
```

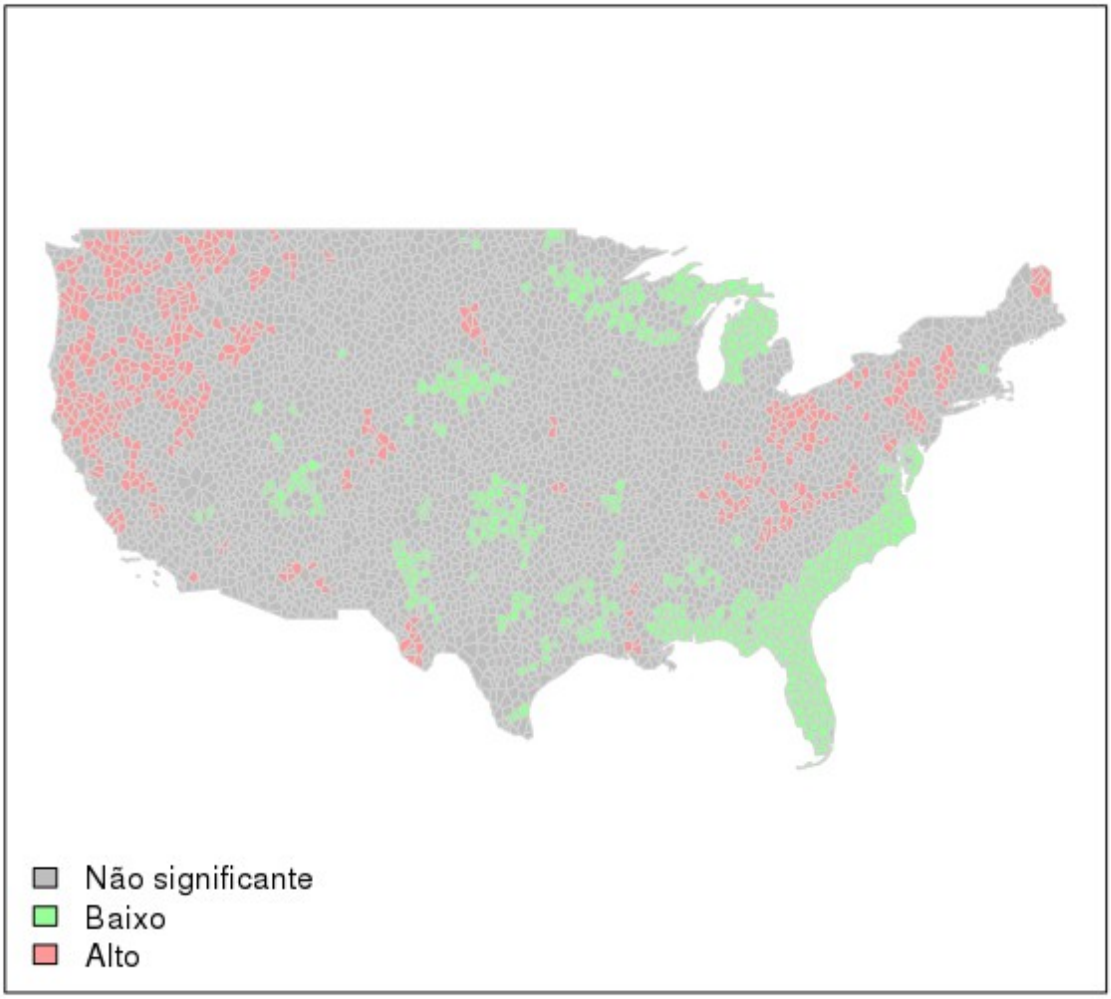
Uma vez delineadas as faixas vamos criar um padrão de cores para cada uma e definir as suas quebras com bases nas faixas definidas acima:

```
> quebras <- c(0,1,2)
> cores <- c('grey',rgb(0,1,0,alpha=0.4),rgb(1,0,0,alpha=0.4))
```

E plotando nosso LISA:

```
> plot(vor.us,border="lightgray",col=cores[findInterval(qua,quebras,all.inside=
FALSE)])
> box()
> legend('bottomleft',legend=c('Não significante','Baixo','Alto'),fill=cores,
bty="n")
```

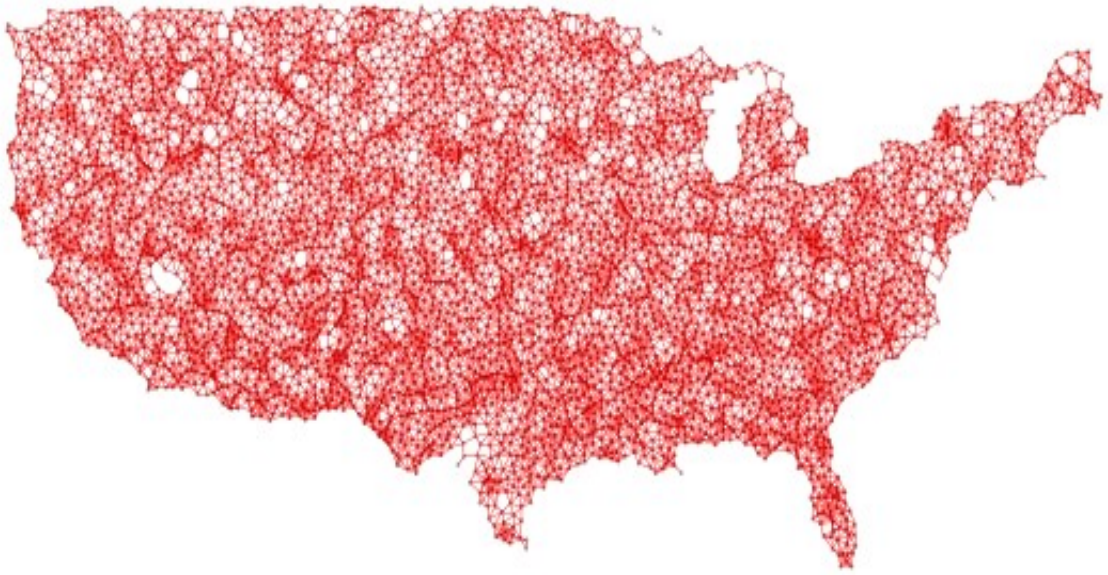
Podemos notar existe um padrão geográfico estatístico significante para a variável teor de zinco do horizonte C dos dados do USGS.



Vamos agora usar o método **Getis-Ord Gi*** nos dados também conhecido com análise de hot-spot. Com esse método achamos zonas quentes no meio de zonas relativamente altas também.

Vamos começar definindo um novo grupo de vizinhos baseados somente na proximidade, não necessariamente dividindo bordas:

```
> vizin2 <- dnearneigh(coordinates(vor.us),0,0.75)
> lista.vz2<-nb2listw(vizin2,style='B')
> plot(vizin2, coordinates(vor.us), col = 'red',lwd=0.2,cex=0.2)
```

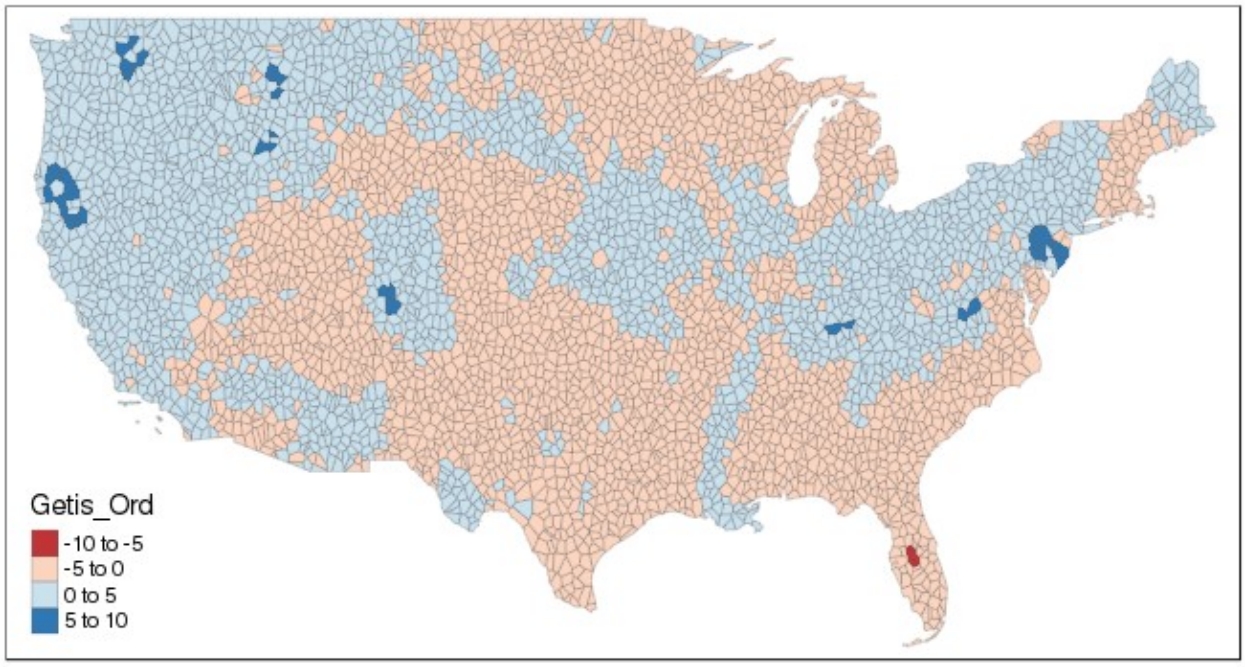


Vamos executar agora o teste a associar o resultado com os nossos dados:

```
> go.local <- localG(vor.us$Zn.ppm, lista.vz2)
> go.local <- cbind(vor.us,as.matrix(go.local))
> names(go.local)[49]<-'Getis_Ord'
```

E plotamos o resultado do teste:

```
> tm_shape(go.local) + tm_fill("Getis_Ord", palette = "RdBu", style = "pretty")
+ tm_borders(alpha=0.3)
```



Localização de minas de Zinco atuais (esquerda) e antigas (direita). Alguma correlação ou alvo potencial?

Fonte: wikipedia.org

Vamos agora efetuar um último teste usando um modelo de regressão linear já visto anteriormente. Usaremos ele para efetuar um teste de **modelo GWR** (Geographically Weighted Regression).

Usaremos o pacote spgwr neste teste. Carregando dados:

```
> geoqui<-read.csv('usa_quim_solo.csv')
> library(raster)
> coordinates(geoqui)<-~Longitude+Latitude
> library(rgdal)
> us.sp<-readOGR(dsn='.',layer='us')
> t.str <- CRS('+init=epsg:4326')
> crs(geoqui)<-CRS('+init=epsg:4269')
> geoqui.sp<-spTransform(geoqui, t.str)
> library(dismo)
> v<-voronoi(geoqui.sp)
> vor.us<-intersect(v,us.sp)
> library(spgwr)
```

Calculando faixas centrais do nosso modelo de regressão linear que será Zn ~ Fe + Sc:

```
> faixa.gwr<-gwr.sel(vor.us$Zn.ppm ~ vor.us$Fe.pct + vor.us$Sc.ppm, data=
vor.us, adapt=TRUE)
Adaptive q: 0.381966 CV score: 4729382
Adaptive q: 0.618034 CV score: 4804653
Adaptive q: 0.236068 CV score: 4633452
Adaptive q: 0.145898 CV score: 4538073
Adaptive q: 0.09016994 CV score: 4448293
Adaptive q: 0.05572809 CV score: 4364782
Adaptive q: 0.03444185 CV score: 4286586
Adaptive q: 0.02128624 CV score: 4214480
Adaptive q: 0.01315562 CV score: 4155333
Adaptive q: 0.008130619 CV score: 4098432
Adaptive q: 0.005024999 CV score: 4043833
Adaptive q: 0.00310562 CV score: 4028998
Adaptive q: 0.002094216 CV score: 4063386
Adaptive q: 0.003793893 CV score: 4030230
Adaptive q: 0.003264269 CV score: 4028113
Adaptive q: 0.003385473 CV score: 4028364
Adaptive q: 0.003304959 CV score: 4028078
Adaptive q: 0.003304959 CV score: 4028078
```

Criando o modelo GWR (demora um pouco, 35 minutos):

```
> modelo.gwr<-gwr(vor.us$Zn.ppm ~ vor.us$Fe.pct + vor.us$Sc.ppm, data=vor.us,
adapt=faixa.gwr,hatmatrix=TRUE,se.fit=TRUE)
```

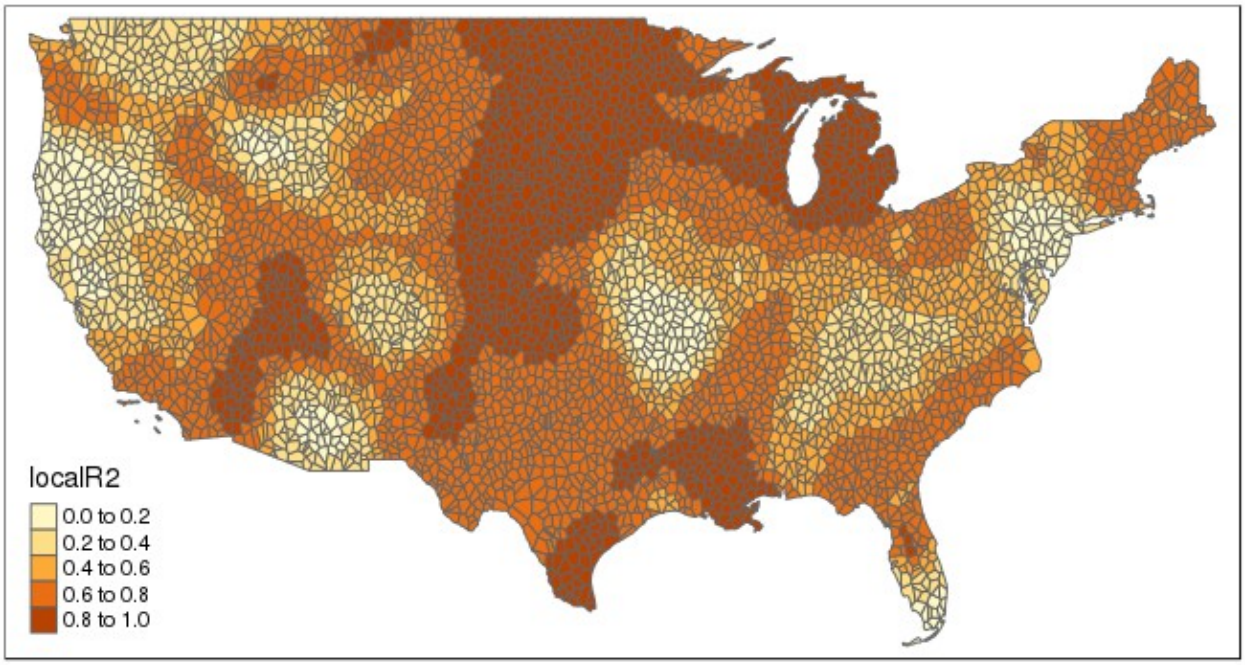
Visualizando as informações do modelo:

```
> modelo.gwr
Call:
gwr(formula = vor.us$Zn.ppm ~ vor.us$Fe.pct + vor.us$Sc.ppm,
    data = vor.us, adapt = faixa.gwr, hatmatrix = TRUE, se.fit = TRUE)
Kernel function: gwr.Gauss
Adaptive quantile: 0.003304959 (about 15 of 4780 data points)
Summary of GWR coefficient estimates at data points:
                    Min.     1st Qu.      Median     3rd Qu.        Max.
X.Intercept.  -50.3891434   1.3359464   7.7870908  22.6321424 124.5397486
vor.us.Fe.pct -29.0557695   4.5653397  10.0581350  17.9216100  46.8575031
vor.us.Sc.ppm -10.9661002   0.0075521   1.7562007   4.3210833  14.9211347
              Global
X.Intercept.  19.9532
vor.us.Fe.pct 12.4088
vor.us.Sc.ppm  0.6649
Number of data points: 4780
Effective number of parameters (residual: 2traceS - traceS'S): 542.7648
Effective degrees of freedom (residual: 2traceS - traceS'S): 4237.235
Sigma (residual: 2traceS - traceS'S): 27.01645
Effective number of parameters (model: traceS): 377.4562
Effective degrees of freedom (model: traceS): 4402.544
Sigma (model: traceS): 26.50439
Sigma (ML): 25.4364
AICc (GWR p. 61, eq 2.33; p. 96, eq. 4.21): 45325.13
AIC (GWR p. 96, eq. 4.22): 44880.4
Residual sum of squares: 3092710
Quasi-global R2: 0.5955108
```

A primeira vista a maioria da informação é bastante similar ao de um modelo de regressão típico, no entanto nos podemos explorar os coeficientes deste modelo em cada unidade de área.

Vamos agora pegar os resultados como um data.frame e unir com os nossos dados:

```
> gwr.res<-as.data.frame(modelo.gwr$SDF)
> names(gwr.res)
 [1] "sum.w"               "X.Intercept."        "vor.us.Fe.pct"
 [4] "vor.us.Sc.ppm"       "X.Intercept._se"     "vor.us.Fe.pct_se"
 [7] "vor.us.Sc.ppm_se"    "gwr.e"               "pred"
[10] "pred.se"             "localR2"             "X.Intercept._se_EDF"
[13] "vor.us.Fe.pct_se_EDF" "vor.us.Sc.ppm_se_EDF" "pred.se.1"
> gwr.final<-cbind(vor.us,as.matrix(gwr.res))
> library(tmap)
> qtm(gwr.final, fill = "localR2")
```



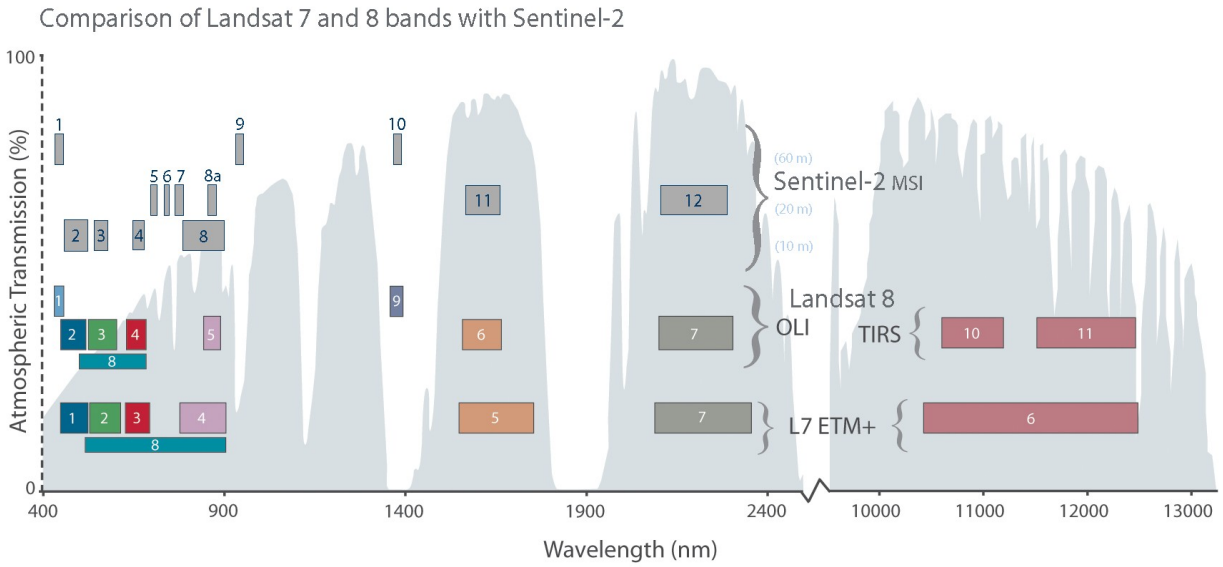
Quanto maior o valor R quadrado, mais explicativo é o modelo para a área específica.

3.2 – Análise espacial de Rasters e imagens de satélite

Vamos agora abordar a análise espacial de imagens do tipo raster, inicialmente vamos ver imagens de satélite multiespectrais e modelos de elevação digital. Posteriormente vamos ver raster em geral na análise espacial de dados geológicos.

3.2.1 - Imagens multiespectrais

As faixas espectrais variam com o tipo de satélite que adquiriu a imagem. Elas se dividem basicamente em visível (azul, verde e vermelho), infravermelho muito próximo (VNIR), infravermelho próximo (NIR), infravermelho de ondas curtas (SWIR) e termal.



Vamos aqui trabalhar com imagens do tipo sentinel2 (cortesia ESA) com as bandas 2, 3, 4, 5, 6, 7, 8, 8A, 11 e 12. Também usaremos DEM de alta resolução do ALOS. Nosso foco será em diferenciação de tipos litológicos e estrutural mas alguns modelos gerais de ambiental (vegetação) e geotecnia (encostas).

As bandas 2, 3 e 4 correspondem à faixa do visível nas cores azul, verde e vermelho respectivamente. Vamos primeiro cria uma composição de cores verdadeiras (true composite color) com as bandas 4, 3 e 2.

Vamos criar também duas composições de cores falsas (false color composite) para as bandas 7, 6 e 5 que representam o infravermelho muito próximo (boas para identificação de vegetação e delineamento estrutural) E outra FCC com as bandas 12,11 e 8A que possui boa sensibilidade para terrenos rochosos e solos.
Criando o TCC :

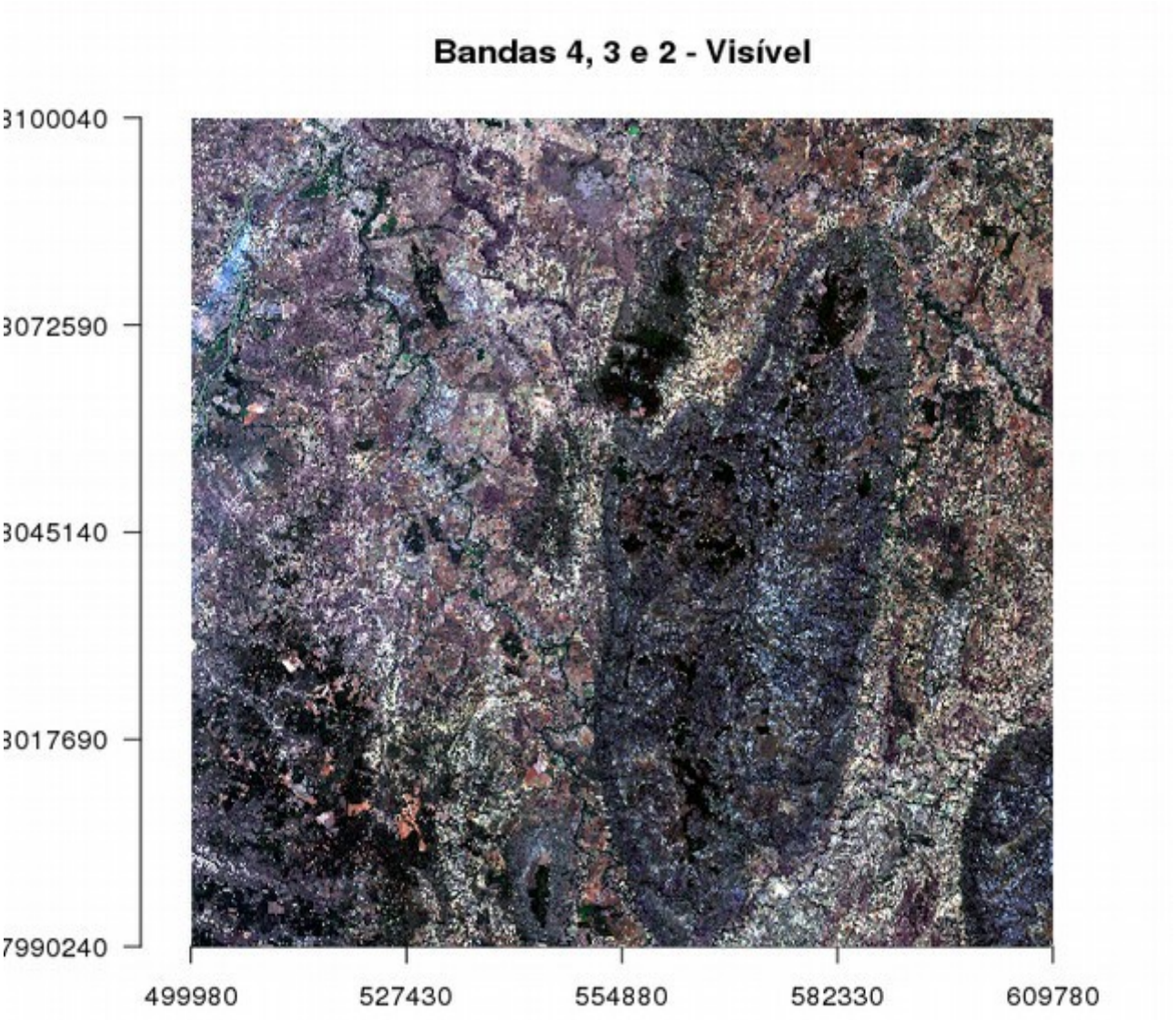
```
> library(raster)
> tcc4_3_2<-stack('b4lr.tif','b3lr.tif','b2lr.tif')
```

Criando os dois FCC:

```
> fcc7_6_5<-stack('b7.tif','b6.tif','b5.tif')
> fcc12_11_8a<-stack('b12.tif','b11.tif','b8a.tif')
```

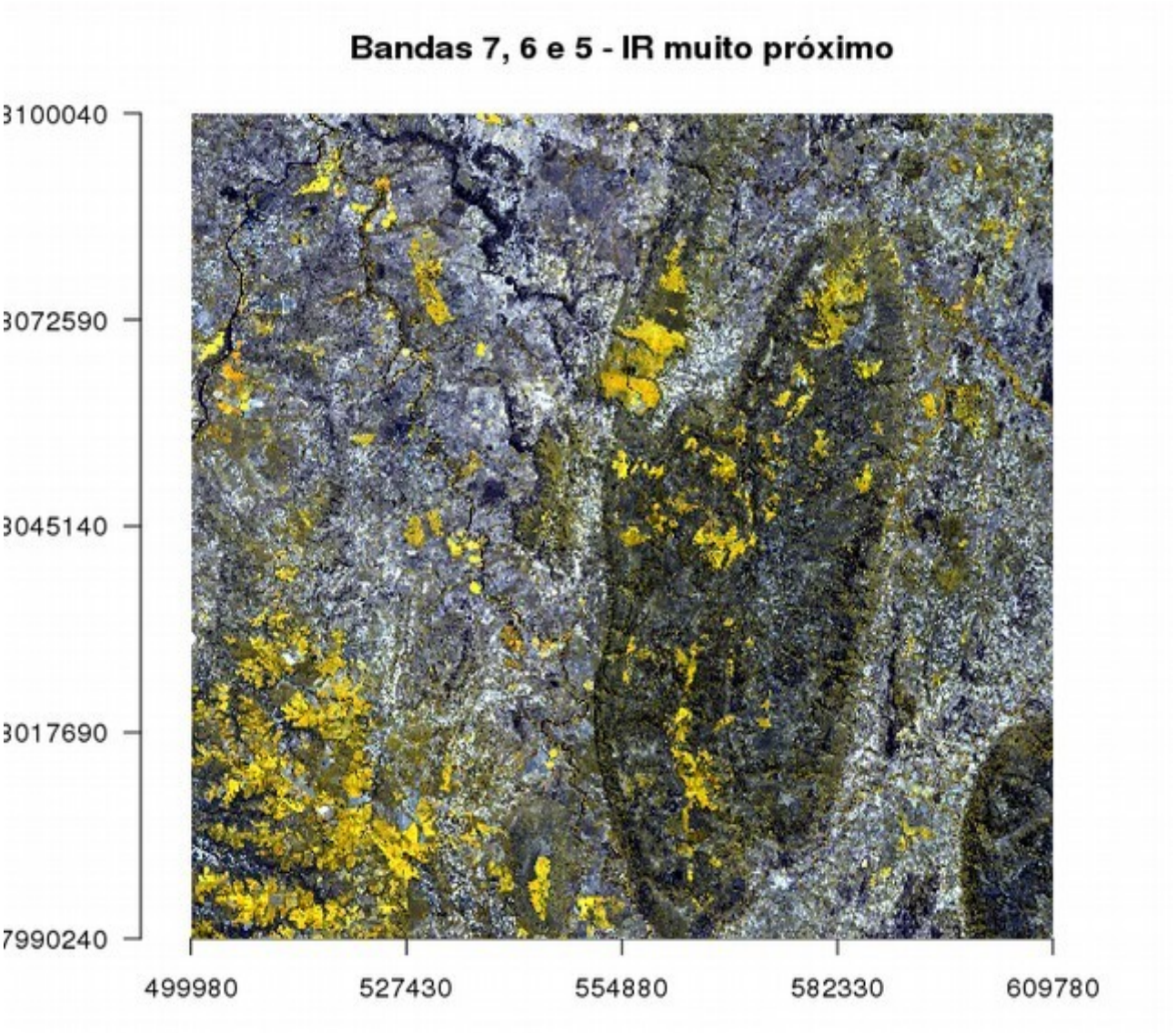
Vamos plotar essas composições coloridas:

```
> plotRGB(tcc4_3_2,stretch='lin',axes=TRUE,main='Bandas 4, 3 e 2 - Visível')
```



Cores naturais, próximas do que o olho veria.

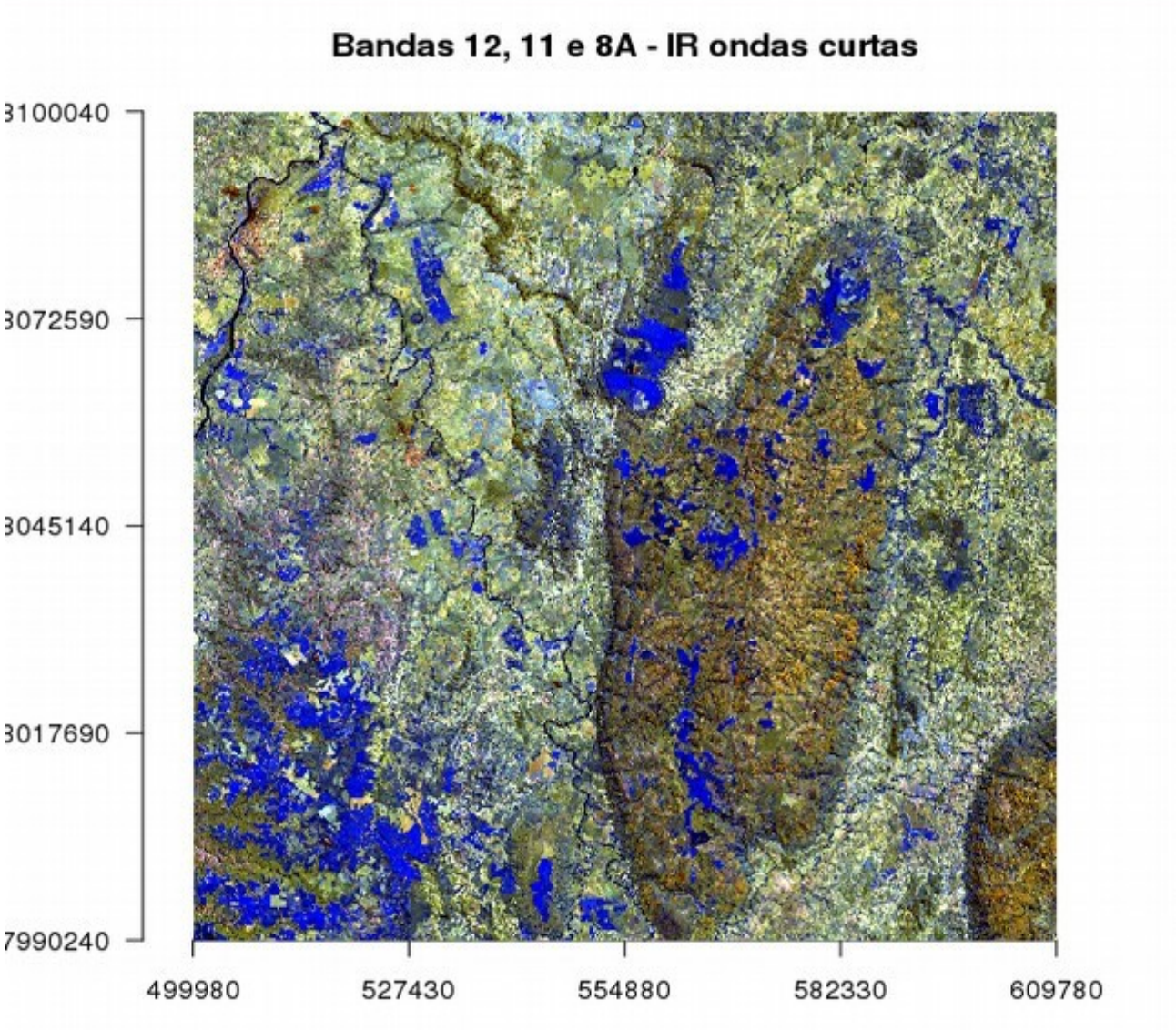
```
> plotRGB(fcc7_6_5,stretch='lin',axes=TRUE,main='Bandas 7, 6 e 5 - IR muito
próximo')
```



Observe a nítida resposta da vegetação saudável e realce de estrutural geológicas.

```
> plotRGB(fcc12_11_8a,stretch='lin',axes=TRUE,main='Bandas 12, 11 e 8A - IR
ondas curtas')
```

Note a diferente respostas do solo e rochas com relação à vegetação.



Trabalhar com imagens de satélite de áreas muito grandes pode ser bastantes penoso para a memória física do computador. Sempre grave as imagens geradas e carregue conforme necessário. Após gravar limpe as variáveis do ambiente R com rm(nome da variável) e use gc().

Gravando as composições e limpando memória:

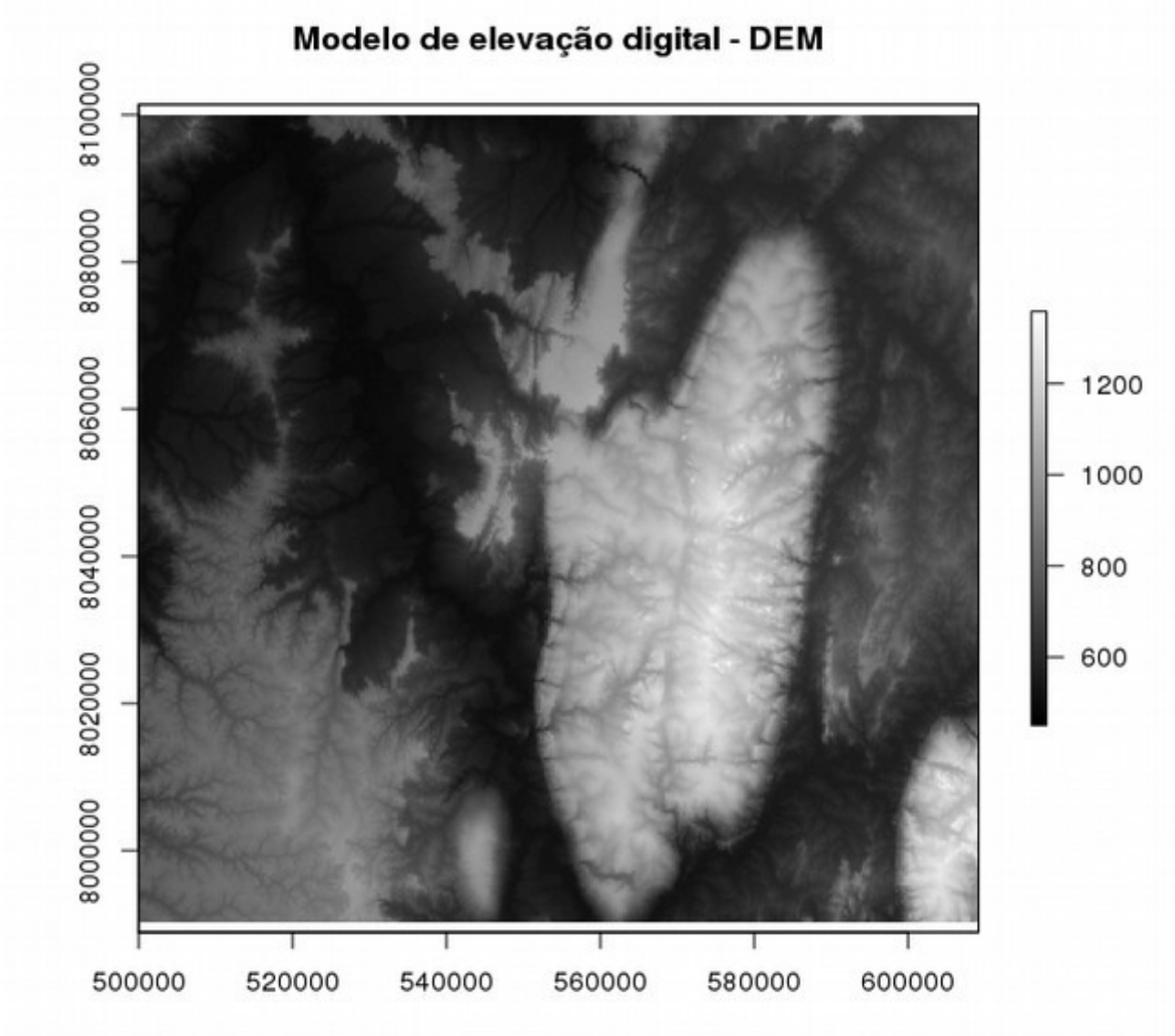
```
> writeRaster(tcc4_3_2, filename='tcc4_3_2.tif', drive='GeoTiff')
> writeRaster(fcc7_6_5, filename='fcc7_5_6.tif', drive='GeoTiff')
> writeRaster(fcc12_11_8a, filename='fcc12_11_8a.tif', drive='GeoTiff')
> rm(list=ls()) #limpa todas as variáveis da sessão
> gc()
```

3.2.2 - Tratamentos de Modelos de Elevação Digital (DEM)

Vamos carregar agora imagem *ALOS World 3D 30m* (cortesia JAXA) da área das imagens Sentinel2.

Carregando e plotando DEM:

```
> library(raster)
> dem<-raster('dem.tif')
> plot(dem,col=grey(0:100/100),main='Modelo de elevação digital - DEM')
```



Podemos criar diversos rasters derivados deste DEM usando:

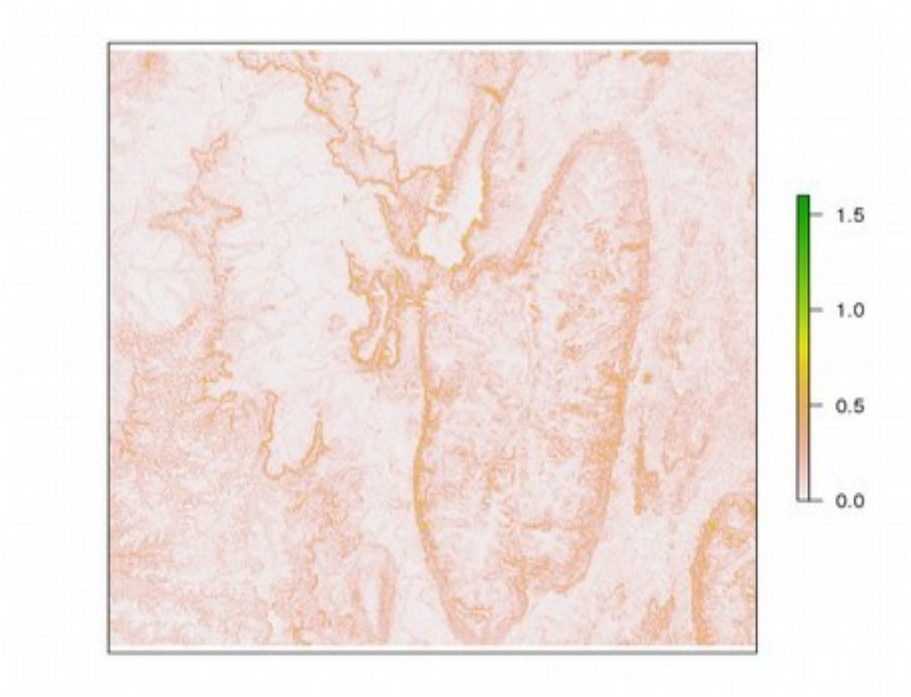
```
> gradiente<-terrain(dem,opt='slope',unit='radians')
> aspecto<-terrain(dem,opt='aspect',unit='radians')
> tpi<-terrain(dem,opt='TPI')
> tri<-terrain(dem,opt='TRI')
> rugosidade<-terrain(dem,opt='roughness')
> fluxo<-terrain(dem,opt='flowdir',unit='radians')
```

Vamos dar uma olhada em cada um desses agora.

Inclinação é o plano que tangencia um determinado ponto na superfície divido em dois parâmetros: Gradiente e aspecto.

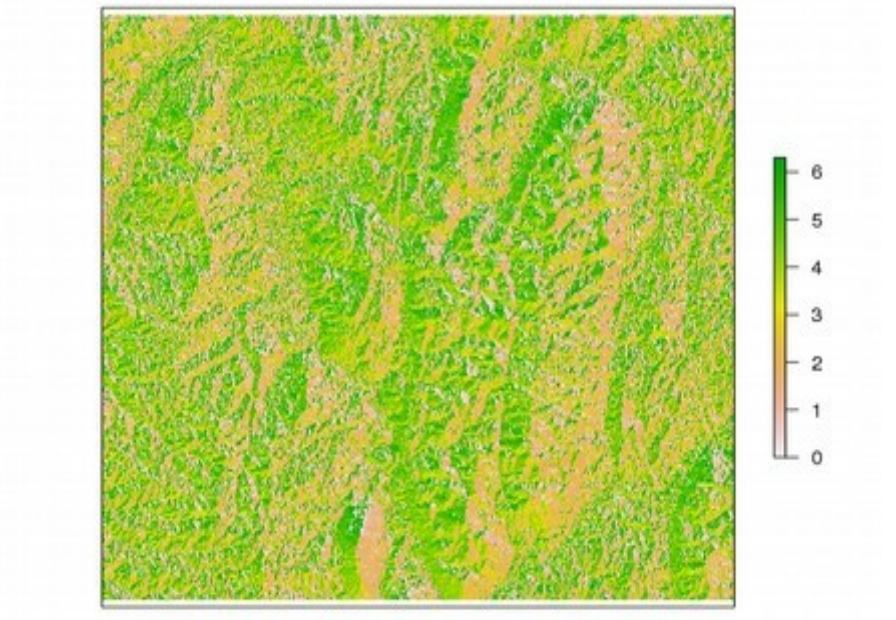
Gradiente ou derivada: ângulo que este plano faz com o plano horizontal (0 a pi/2 em radianos):

```
> plot(gradiente,zlim=c(0,1.6),axes=FALSE)
```



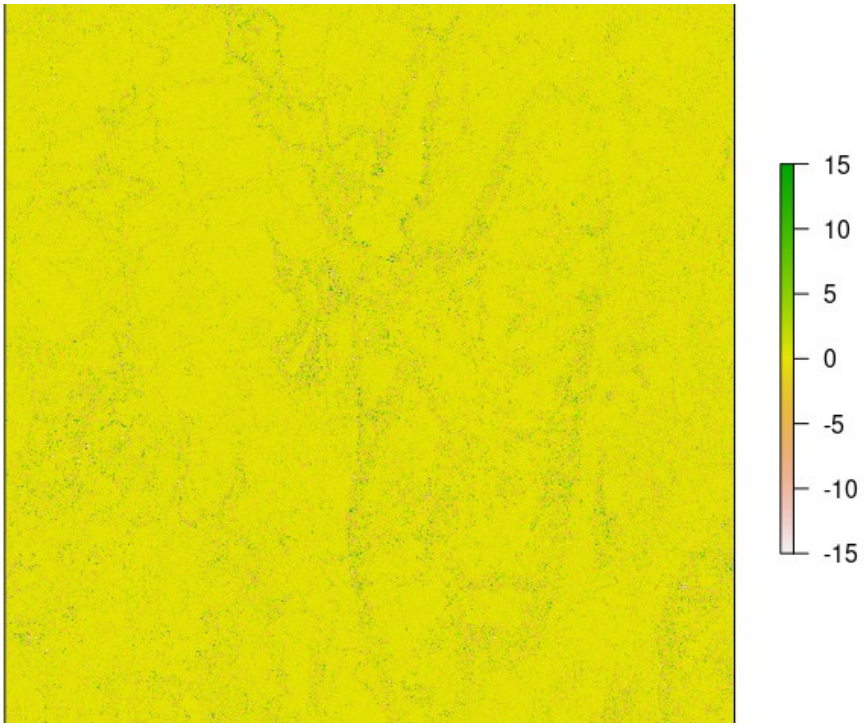
Aspecto: a direção do mergulho máximo deste plano com relação a 0, geralmente Norte (0 a 2pi):

```
> plot(aspecto,zlim=c(0,6.3),axes=FALSE)
```



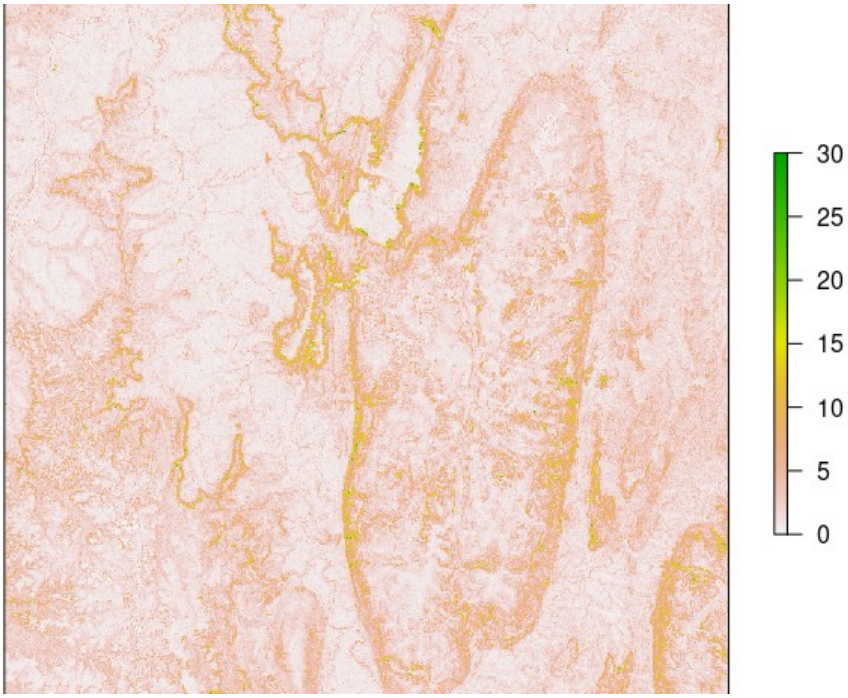
TPI: índice de posição topográfica, é a diferença da altitude de cada célula em um DEM com a elevação média de sua vizinhança, valores positivos representam células que são mais altas do que a média da vizinhança, Valores próximo a zero são áreas planas ou de gradiente constante:

```
> plot(tpi,zlim=c(-15,15),axes=FALSE)
```



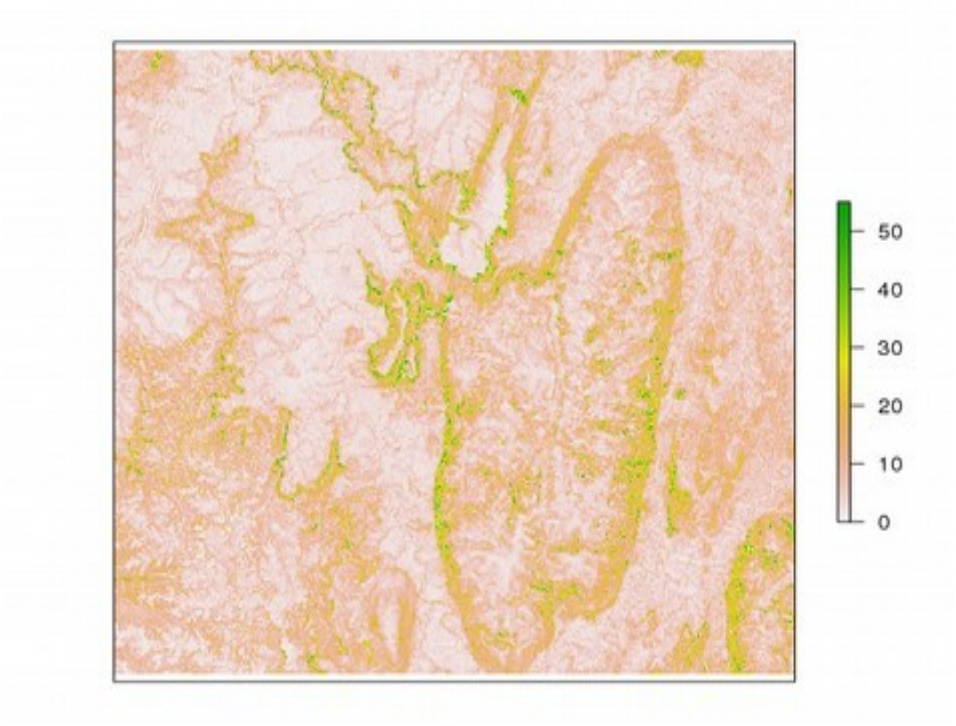
TRI: índice de rugosidade topográfica expressa a média da diferença absoluta de elevação entre as células adjacentes com esta célula:

```
> plot(tri,zlim=c(0,30),axes=FALSE)
```



Rugosidade: é a diferença entre o valor máximo e o mínimo de uma célula e suas oito células vizinhas:

```
> plot(rugosidade,zlim=c(0,55),axes=FALSE)
```

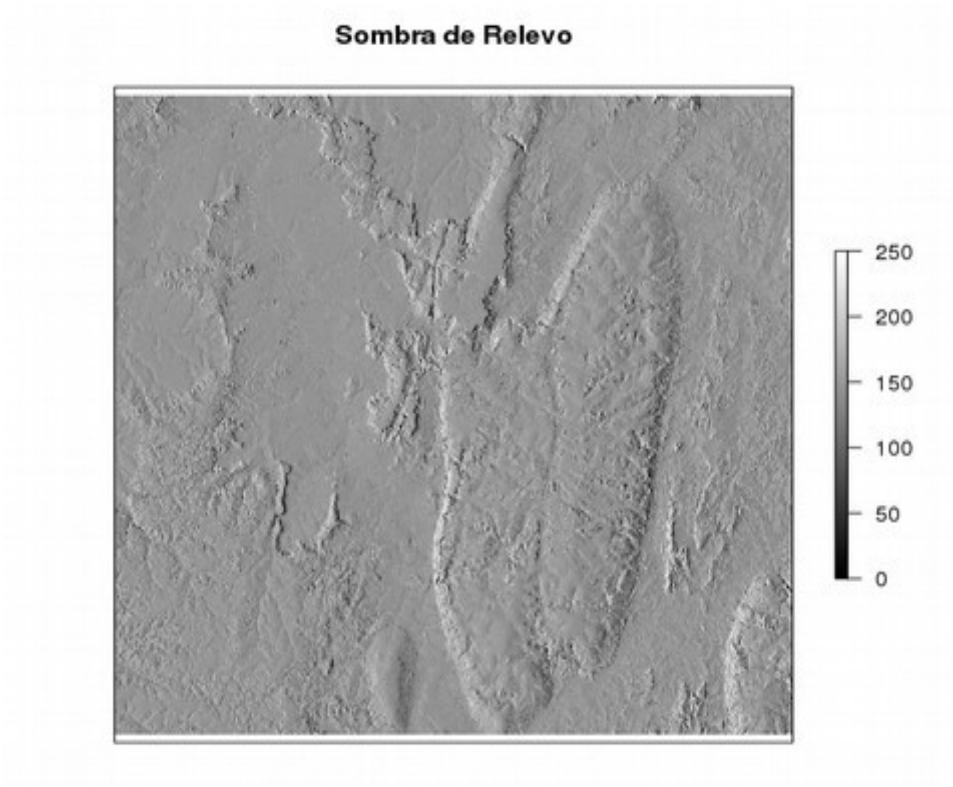


Fluxo: realça a direção de fluxo gravitacional líquido i.e. a direção de maior queda em elevação ou de menor aclive no caso de todos os vizinhos serem mais alto:

```
> plot(fluxo,zlim=c(0,20),axes=FALSE)
```

Vamos agora criar uma sombra de relevo com do nosso modelo digital de elevação usando os dados de inclinação e aspecto extraídos acima:

```
> relevo<-hillShade(gradiente,aspecto,angle=45,direction=270,normalize= TRUE)
> plot(relevo,col=grey(0:100/100),zlim=c(0,250),main='Sombra de Relevo')
```



Vamos gravar nossa sombra de relevo num arquivo chamado hillshade.tif:

```
> writeRaster(relevo, filename='hillshade.tif', drive='GeoTiff')
```

3.2.3 - Tratamentos de imagens multiespectrais

Vamos agora ver dois tratamentos muito usados em processamento de imagem raster (principalmente para imagens multiespectrais de satélite). Elas são Análise de Componentes Principais (PCA) e transformação IHS.

Vamos começar pela **Análise de Componentes Principais (PCA).** Primeiro carregamos todas a bandas (com mesma resolução) que trabalharemos a análise. Vamos carregar as bandas 2,3,4 em baixa resolução e as bandas 5, 6, 7, 8A, 11 e 12 dentro de um objeto Stack.

```
> library(raster)
> rlis<-list("b2lr.tif","b3lr.tif","b4lr.tif","b5.tif","b6.tif","b7.tif",
"b8a.tif","b11.tif","b12.tif")
> sentinel<-stack(rlis)
```

O próximo passo é criar uma amostragem aleatória de vários pontos deste conjunto de imagens:

```
> set.seed(1)
> amostra <- sampleRandom(sentinel, 300000) # demora uns 20 minutos
```

Criamos o nosso PCA com a função prcomp e em seguida removemos a varável amostra para liberar memória:

```
> pca <- prcomp(amostra, scale = TRUE)
> rm(amostra)
```

Dando uma olhada na nossa análise:

```
> pca
Standard deviations (1, .., p=9):
[1] 2.4971707 1.4360686 0.6518079 0.3255979 0.2417147 0.2219534 0.1722686
[8] 0.1367066 0.1221613

Rotation (n x k) = (9 x 9):
            PC1         PC2          PC3         PC4         PC5         PC6
b2lr -0.3491652  0.22185994 -0.480746376  0.49014513 -0.31494305 -0.2952344
b3lr -0.3691301  0.15139738 -0.444995192 -0.01065041 -0.01253432  0.2066793
b4lr -0.3698774  0.19362301 -0.239081751 -0.35828268  0.63678708  0.2458780
b5   -0.3831135  0.07436419  0.172037720 -0.60845361 -0.28449066 -0.5693550
b6   -0.3019818 -0.44541792  0.019527675 -0.19438767 -0.30523645  0.2549897
b7   -0.2546944 -0.53162958 -0.009767061  0.04788285 -0.06777289  0.2385668
b8a  -0.2500379 -0.53071467  0.049005000  0.31416451  0.35077891 -0.3303194
b11  -0.3506655  0.21081265  0.525330502  0.30093007  0.32087614 -0.1780180
b12  -0.3416047  0.27876499  0.453050536  0.17811362 -0.30075116  0.4719789
             PC7         PC8         PC9
b2lr  0.34493317 -0.2174652 -0.06985700
b3lr -0.71347752  0.2975117  0.04744295
b4lr  0.39849832 -0.1207889  0.02880283
b5    0.03385582  0.1960893 -0.05642686
b6   -0.11140726 -0.6334559  0.31682006
b7    0.14016855  0.2240491 -0.72025001
b8a   0.04449279  0.3209524  0.46644966
b11  -0.33593702 -0.3683746 -0.28784582
b12   0.25520917  0.3478218  0.26323003
```

E criando imagens de cada componente usando:

```
> pci1<-predict(sentinel,pca,index=1)
> pci2<-predict(sentinel,pca,index=2)
> pci3<-predict(sentinel,pca,index=3)
> pci4<-predict(sentinel,pca,index=4)
> pci5<-predict(sentinel,pca,index=5)
> pci6<-predict(sentinel,pca,index=6)
> pci7<-predict(sentinel,pca,index=7)
> pci8<-predict(sentinel,pca,index=8)
> pci9<-predict(sentinel,pca,index=9)
```

E agora gravando cada componente em um arquivo raster (e liberando memória):

```
> writeRaster(pci1,filename='pci1.tif',driver='Geotiff')
> writeRaster(pci2,filename='pci2.tif',driver='Geotiff')
> writeRaster(pci3,filename='pci3.tif',driver='Geotiff')
> rm(pci1,pci2,pci3,pca) #liberando memoria
> writeRaster(pci4,filename='pci4.tif',driver='Geotiff')
> writeRaster(pci5,filename='pci5.tif',driver='Geotiff')
> writeRaster(pci6,filename='pci6.tif',driver='Geotiff')
> rm(pci4,pci5,pci6)
> writeRaster(pci7,filename='pci7.tif',driver='Geotiff')
> writeRaster(pci8,filename='pci8.tif',driver='Geotiff')
> writeRaster(pci9,filename='pci9.tif',driver='Geotiff')
> rm(pci7,pci8,pci9)
```

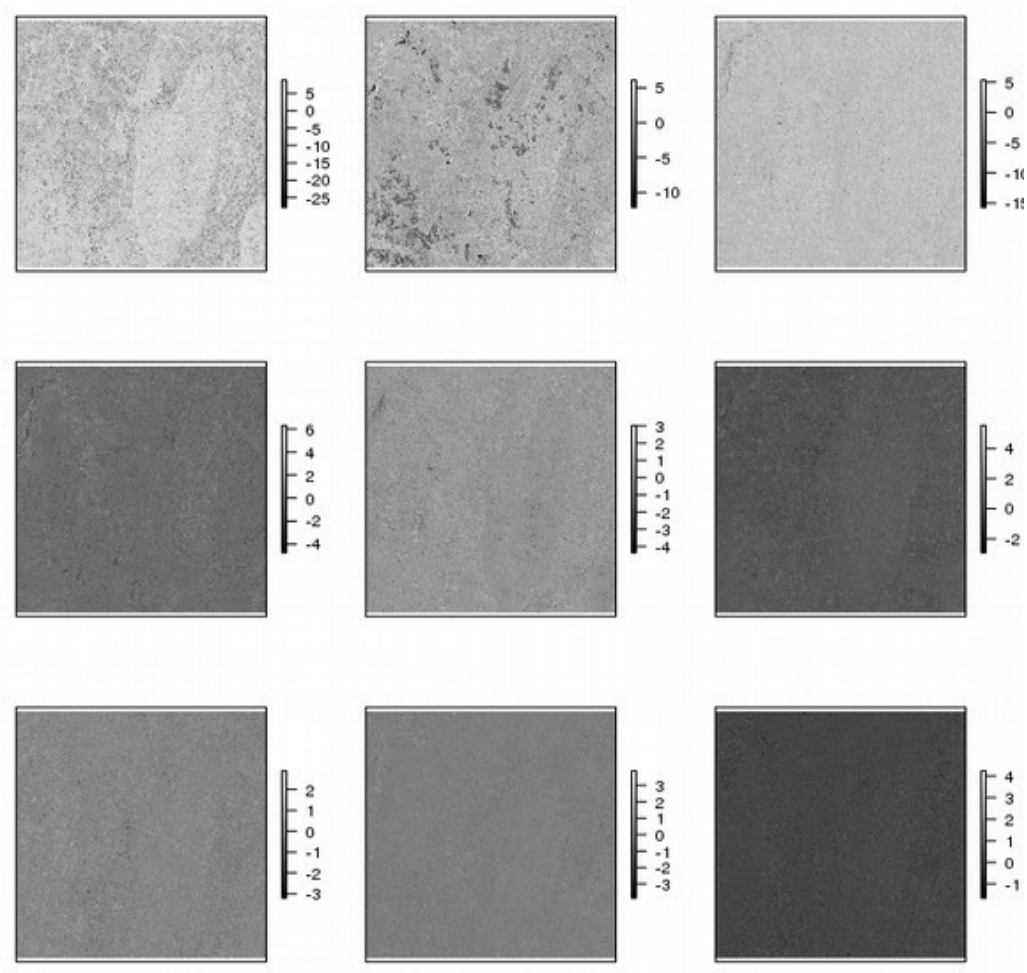
Vamos dar uma olhada nos rasters de cada componente:

```
> library(raster)
> pc1<-raster('pci1.tif')
> pc2<-raster('pci2.tif')
> pc3<-raster('pci3.tif')
> pc4<-raster('pci4.tif')
```

```
> pc5<-raster('pci5.tif')
> pc6<-raster('pci6.tif')
> pc7<-raster('pci7.tif')
> pc8<-raster('pci8.tif')
> pc9<-raster('pci9.tif')
> par(mfrow=c(3,3))
> plot(pc1,axes=FALSE,col=grey(0:100/100))
> plot(pc2,axes=FALSE,col=grey(0:100/100))
> plot(pc3,axes=FALSE,col=grey(0:100/100))
> plot(pc4,axes=FALSE,col=grey(0:100/100))
> plot(pc5,axes=FALSE,col=grey(0:100/100))
> plot(pc6,axes=FALSE,col=grey(0:100/100))
> plot(pc7,axes=FALSE,col=grey(0:100/100))
> plot(pc8,axes=FALSE,col=grey(0:100/100))
> plot(pc9,axes=FALSE,col=grey(0:100/100))
```

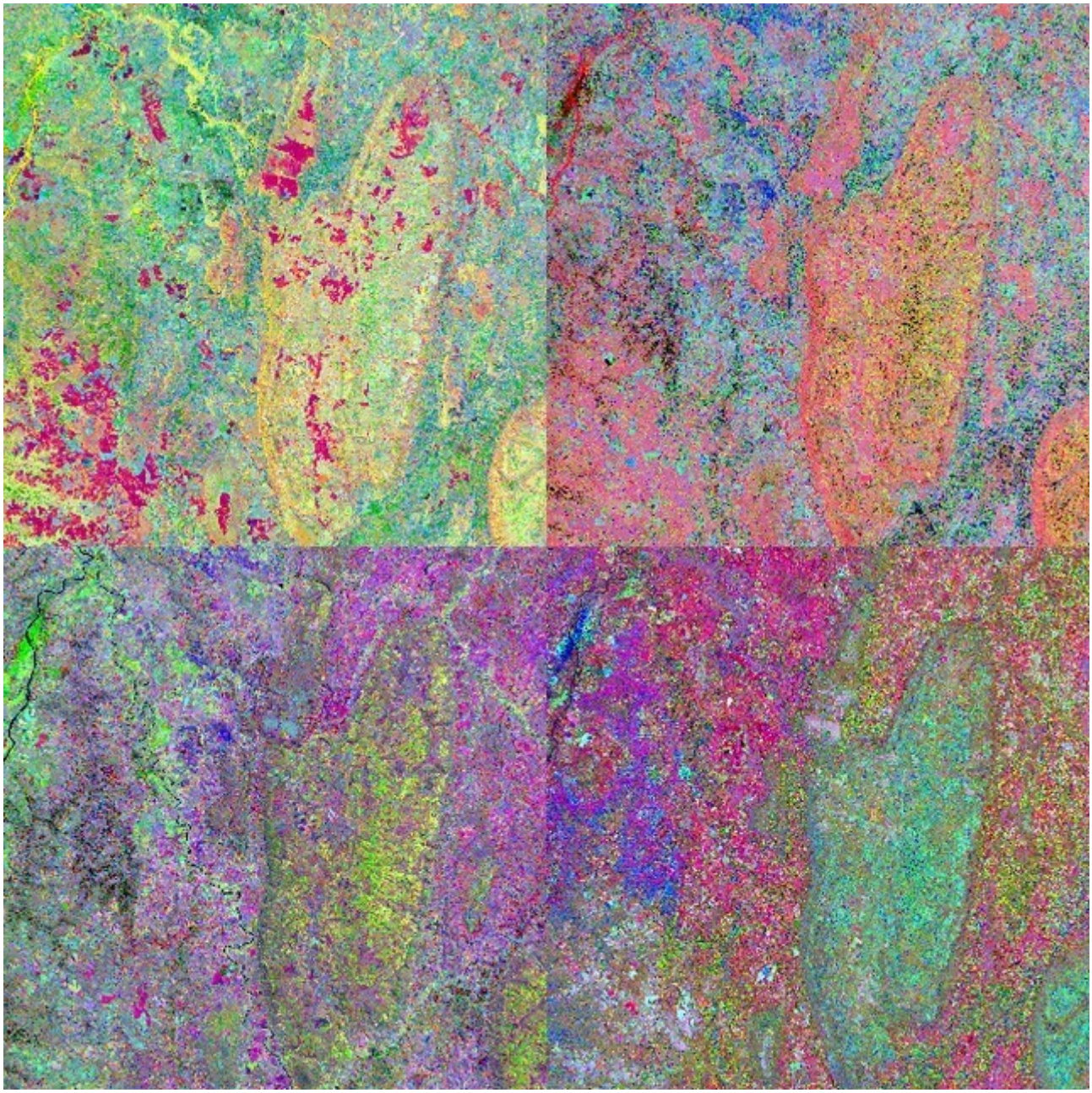
Processamos 9 raster de entrada e geramos através da análise de componentes principais 9 raster de saída ranqueados de forma decrescente de 1 a 9 de acordo com a ordem de suas variâncias (contraste). PC1 tem o maior ranque de importância e PC9 o menor.

Algum dos componentes tem respostas diferentes a certos comprimento de onda uns realçando vegetação natural, outros agricultura, alguns solo e alguns rochas. A combinação destes componentes em composições coloridas falsas (FCC) podem ter ótimos resultados em diferenciar respostas espectrais distintas que estavam pouco perceptíveis nas bandas originais da imagem de satélite.



Criando algumas composições coloridas com os Componentes Principais (com permissão de Andy Warhol):

```
> pc123<-stack(pc1,pc2,pc3)
> pc135<-stack(pc1,pc3,pc5)
> pc345<-stack(pc3,pc4,pc5)
> pc567<-stack(pc5,pc6,pc7)
> par(mfrow=c(2,2))
> plotRGB(pc123,stretch='lin')
> plotRGB(pc135,stretch='lin')
> plotRGB(pc345,stretch='lin')
> plotRGB(pc567,stretch='lin')
```



Veremos agora como é feito uma decomposição **IHS** a partir de um conjunto RGB. Sempre usaremos uma composição RGB de partida para a transformação IHS.
Inicialmente vamos usar a composição fcc12_11_8A como exemplo:

```
> library(raster)
> fcc.rgb<-brick('fcc12_11_8a.tif')
```

Os dados nesse raster são do tipo inteiro 16 bits com valores entre 0 e 65535. Para efetuar a transformação precisamos normalizar estes valores para um faixa entre 0 e 1. Isto é alcançado facilmente dividindo o nosso objeto raster por 65535.

Normalizando os valores para a faixa entre 0 e 1

```
> Nor<-fcc.rgb/65535
> Nor
class       : RasterBrick
dimensions  : 5490, 5490, 30140100, 3  (nrow, ncol, ncell, nlayers)
resolution  : 20, 20  (x, y)
extent      : 499980, 609780, 7990240, 8100040  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=utm +zone=23 +south +datum=WGS84 +units=m +no_defs
+ellps=WGS84 +towgs84=0,0,0
data source : in memory
names       : fcc12_11_8a.1, fcc12_11_8a.2, fcc12_11_8a.3
min values  :             0,             0,             0
max values  :     0.3813687,     0.1921111,     0.1226825
```

Agora criamos cada um dos componente usando as seguintes formulas :

$$I = (R+G+B)/3$$

$$S = 1-(3/(R+G+B)) * minimo(R,G,B)$$

$$H^* = acos((0.5*((R-G)+(R-B))) / (sqrt ((R-B)^2 + ((R-G)*(G-B)))))$$

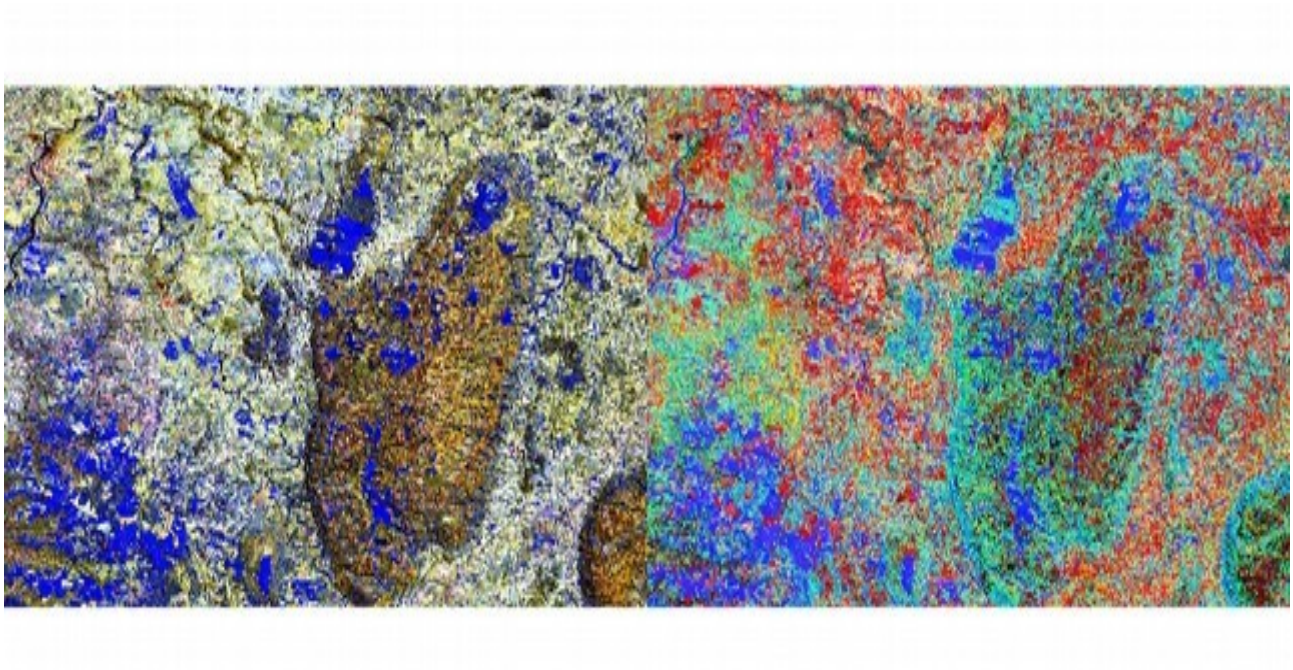
```
> I<-(Nor[[1]]+Nor[[2]]+Nor[[3]])/3
> S<-1-(3/(Nor[[1]]+Nor[[2]]+Nor[[3]]))*min(Nor[[1]],Nor[[2]],Nor[[3]])
> H<- acos((0.5*((Nor[[1]]-Nor[[2]])+(Nor[[1]]-Nor[[3]])))/(sqrt((Nor[[1]]-
Nor[[2]])^2+((Nor[[1]]-Nor[[3]])*(Nor[[2]]-Nor[[3]])))))
```

Criando uma composição IHS colorida:

```
> ihs<-brick(I,H,S)
```

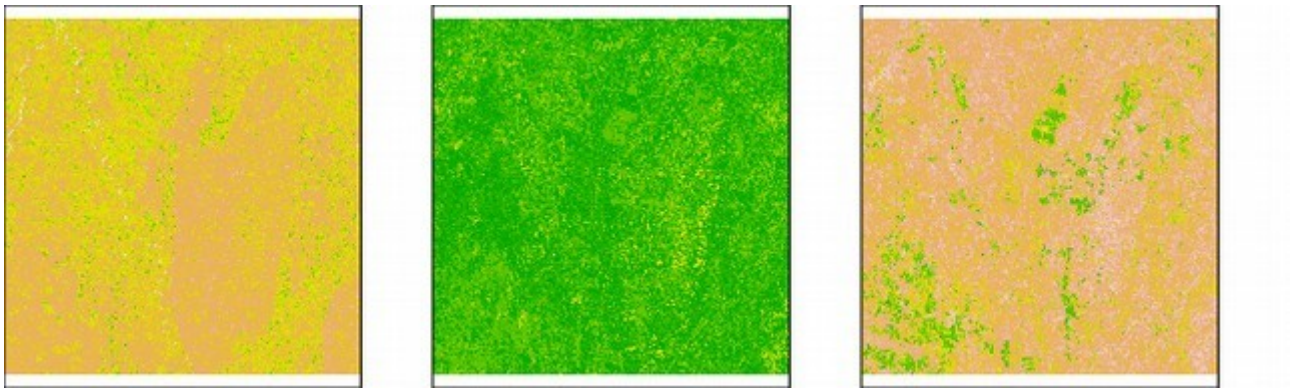
Plotando a FCC original e do IHS:

```
> par(mfrow=c(1,2))
> plotRGB(fcc.rgb,stretch='hist')
> plotRGB(ihs,stretch='hist')
```



Plotando individualmente I, H e S:

```
> par(mfrow=c(1,3))
> plot(I,axes=FALSE,legend=FALSE)
> plot(H,axes=FALSE,legend=FALSE)
> plot(S,axes=FALSE,legend=FALSE)
```



Gravando cada um deles em arquivo Geotiff:

```
> writeRaster(I,filename='I.tif',driver='Geotiff')
> writeRaster(H,filename='H.tif',driver='Geotiff')
> writeRaster(S,filename='S.tif',driver='Geotiff')
```

3.2.4 - Raster no contexto de análise de dados geológicos

Vimos na sessão anterior como gerar novas informações a partir de imagens de satélite. Vamos agora usar esta informação para avaliar e gerar dados geológicos, inicialmente vamos focar em gerar mapas litológicos e estruturais. Em seguida vamos usar dados raster oriundos de levantamentos geofísicos.

Razões entre bandas são constantemente usadas para criar produtos com realce em determinadas faixas espectrais. Queremos realçar respostas oriundas de solos e rochas e em segundo plano, vegetação natural que tendem ocasionalmente responder a determinado tipo de solo.

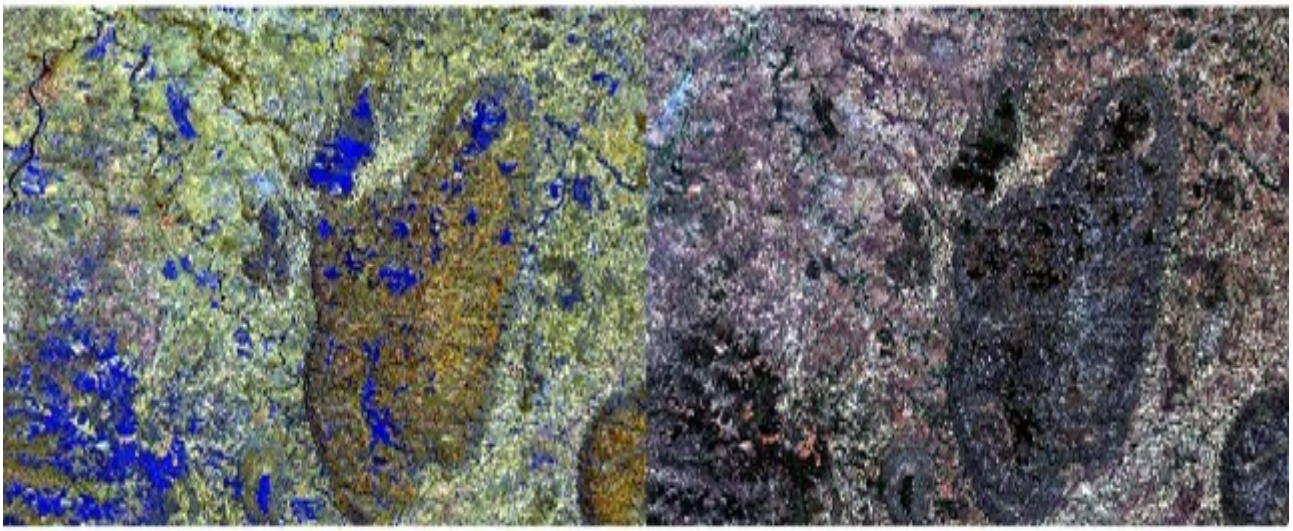
Vamos criar a primeira razão entre composições coloridas que geram uma boa resposta do ponto de vista geológico. A razão de SWIR pelo visível, ou seja, Bandas 12,11, e 8A divididas pelas bandas 4, 3 e 2 nos canais RGB respectivamente, ou seja, R=12/4, G=11/3 e B=8A/2.

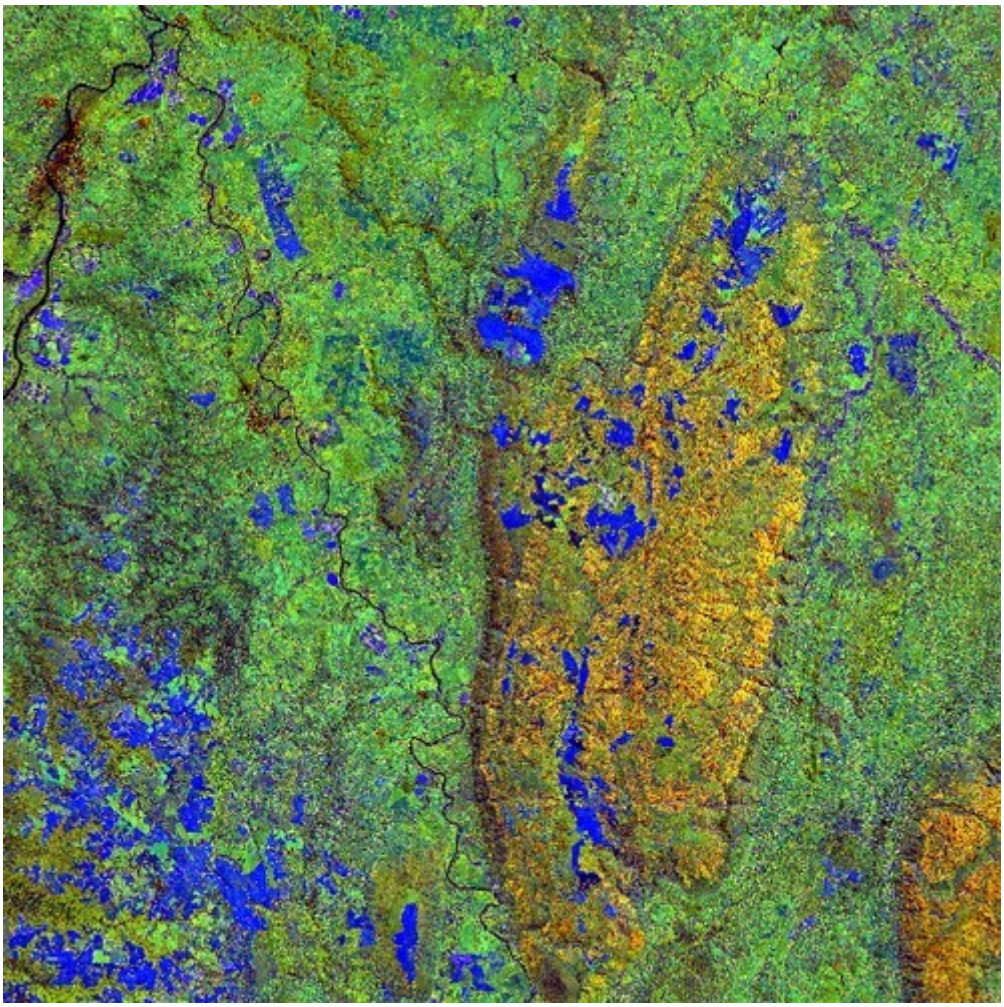
Criando nossa razão entre SWIR e visível:

```
> library(raster)
> swir<-brick('fcc12_11_8a.tif')
> vis<-brick('tcc4_3_2.tif')
> razao<-swir/vis
```

Plotando as composições SWIR, visível e razão SWIR/visível:

```
> par(mfrow=c(1,2))
> plotRGB(swir,axes=FALSE,stretch='lin')
> plotRGB(vis,axes=FALSE,stretch='lin')
> dev.off()
> plotRGB(razao,axes=FALSE,stretch='lin')
```





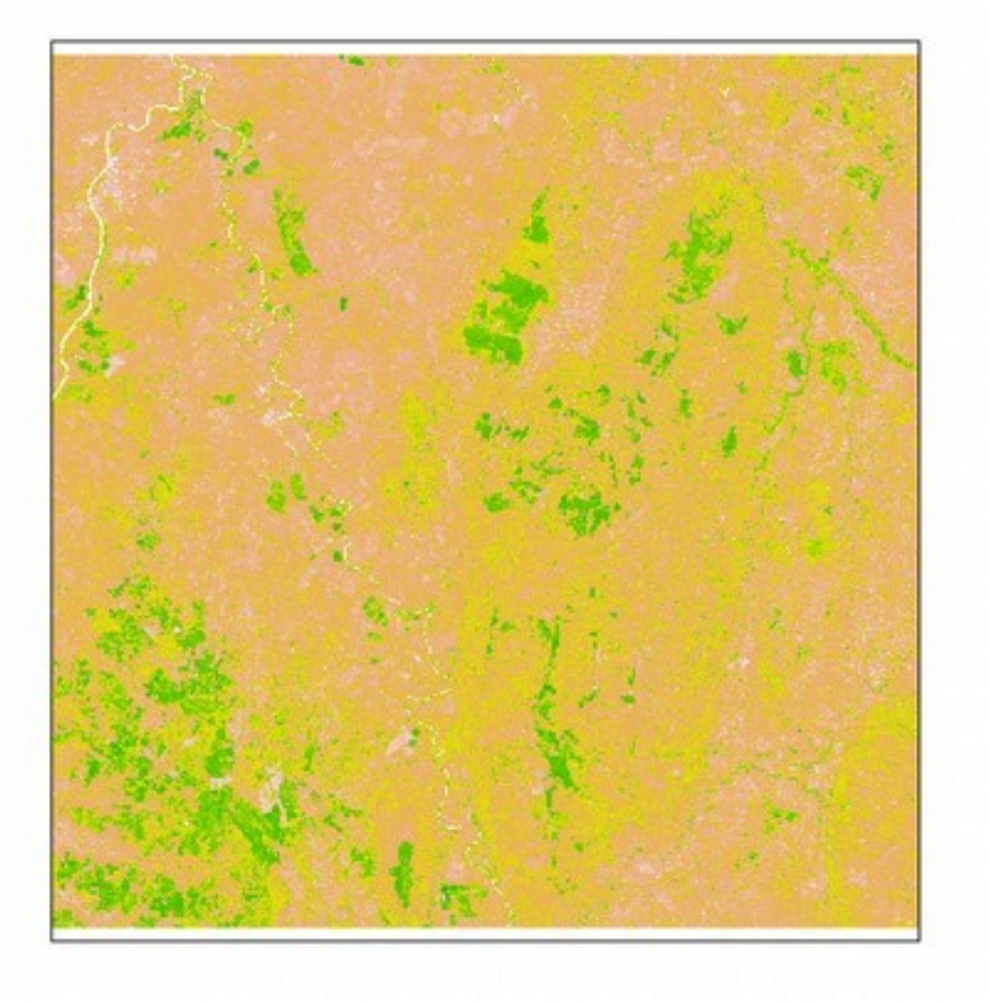
Operações envolvendo bandas são bastante usadas. Uma operação que envolve razão de bandas e bastante usado é o índice da diferença normalizado de vegetação (NDVI). Esse índice reflete a reflectância da clorofila presente nas plantas. Plantas saudáveis têm índice maior, plantas secas, com baixa densidade ou mortas mostram índices menores.

Ele utiliza a banda do infravermelho próximo (banda 8 do sendtinel2) e a banda visível do vermelho (banda 4). A fórmula do NDVI é:

NDVI = (NIR - Vermelho)/(NIR + Vermelho)

Vamos calcular e plotar o NDVI, tons de verde representam índices maiores:

```
> library(raster)
> nir<-raster('b8a.tif')
> vermelho<-raster('b4lr.tif')
> ndvi<-(nir-vermelho)/(nir+vermelho)
> plot(ndvi,axes=FALSE,legend=FALSE,zlim=c(0,1))
```



Embora este índice tenha mais ênfase ambiental e biológica e pode ser bastante útil assinalando áreas de forte vegetação que mascarariam respostas espectrais de solo e rochas. Principalmente se forem culturas antrópicas.

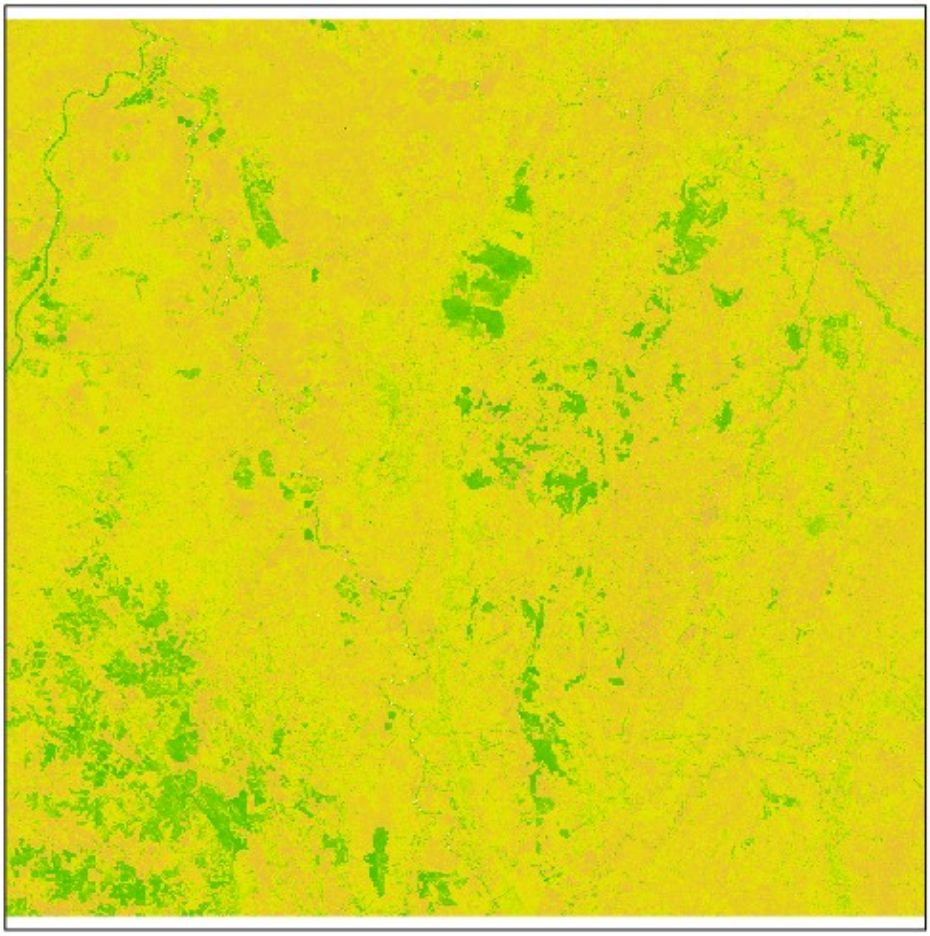
Existem outros índices na literatura mas o princípio é o mesmo. Realçar uma resposta espectral específica. Vamos ver o NDMI (índice da diferença normalizado de umidade) e o NDWI (índice da diferença normalizado de água).

O NDMI utiliza a banda do infravermelho de ondas curtas (banda 11 do sentinel2) e a banda infravermelho próximo (banda 8a do sendtinel2). A fórmula do NDMI é:

NDMI = (NIR-SWIR)/(NIR+SWIR)

Vamos calcular e plotar o NDMI, tons de verde representam índices maiores:

```
> library(raster)
> nir<-raster('b8a.tif')
> swir<-raster('b11.tif')
> ndmi<-(nir-swir)/(nir+swir)
> plot(ndmi,axes=FALSE,legend=FALSE,zlim=c(-1,1))
```



O NDWI utiliza a banda do infravermelho muito próximo (banda 5 do sentinel2) e a banda do verde (banda 3 do sendtinel2). A fórmula do NDWI é:

NDWI = (GREEN-VNIR)/(GREEN+VNIR)

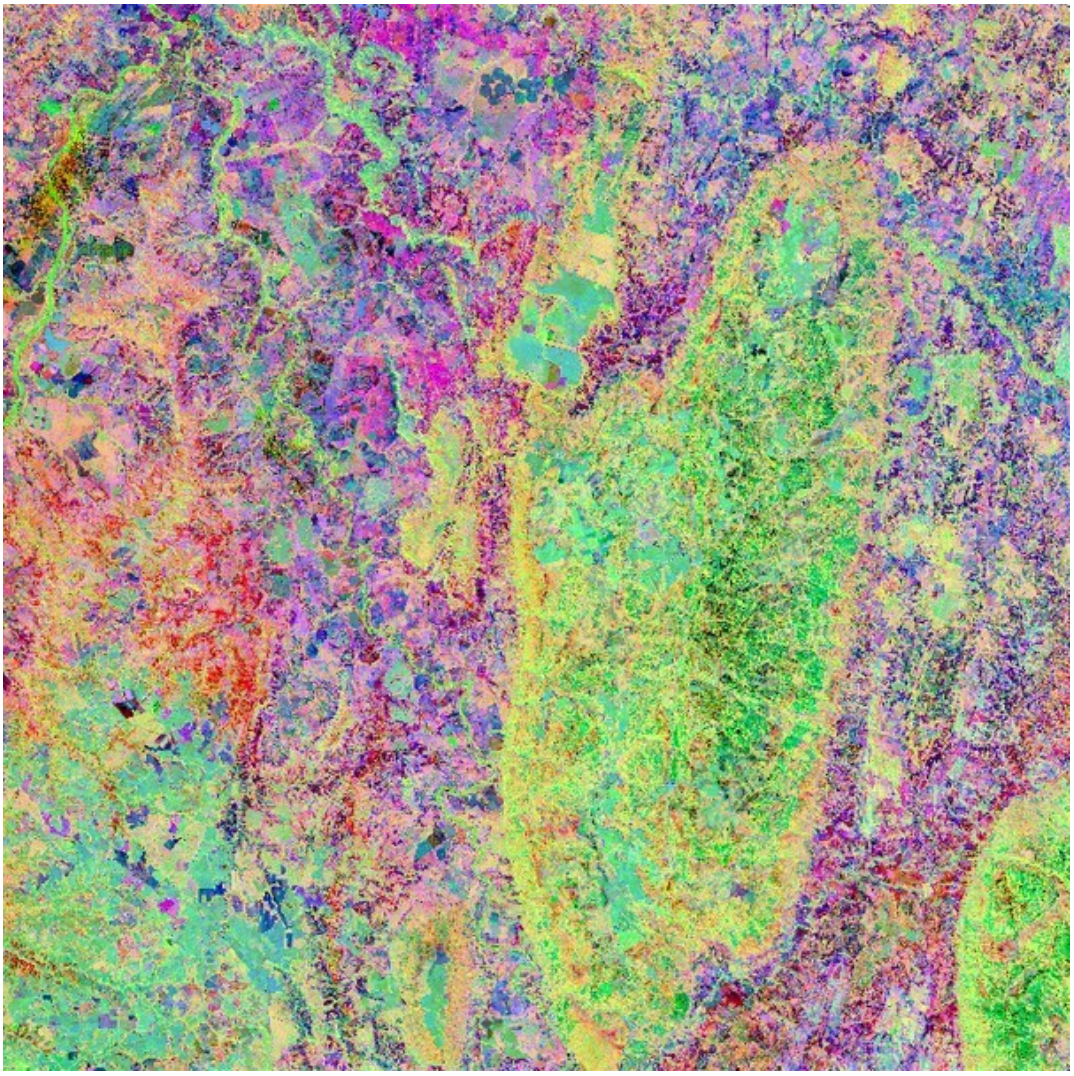
Vamos calcular e plotar o NDMI, tons de verde representam índices maiores:

```
> library(raster)
> green<-raster('b3lr.tif')
> vnir<-raster('b5.tif')
> ndwi<-(green-vnir)/(green+vnir)
> plot(ndwi,axes=FALSE,legend=FALSE,zlim=c(0,1))
```

Agora vamos integrar dados de PCA com dados de IHS para tentar realçar diferenças litológicas. Na maioria das vezes se trata de uma escolha arbitrária de qual composição se comporta melhor mas algumas composições têm melhores respostas comprovadas na literatura.

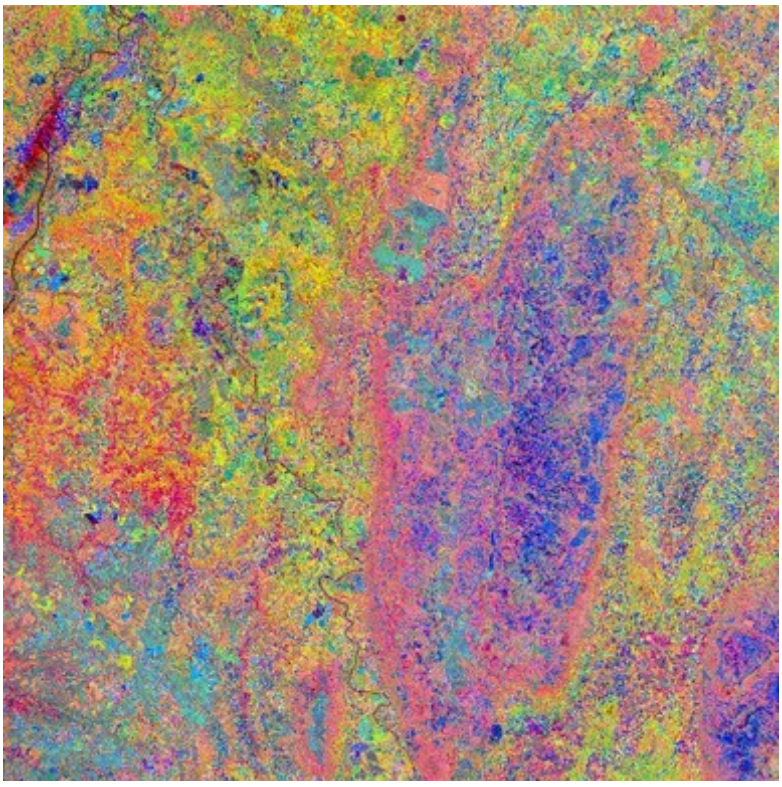
Já criamos acima uma composição com boa resposta (razão SWIR/visível), vamos criar alguma usando PCA e IHS.
H, PC1, PC5:

```
> library(raster)
Carregando pacotes exigidos: sp
> ihs<-stack('I.tif','H.tif','S.tif')
> pca<-stack('pci1.tif','pci2.tif','pci3.tif','pci4.tif','pci5.tif','pci6.tif',
'pci7.tif','pci8.tif','pci9.tif> pca
> H_PC1_PC5<-stack(ihs[[2]],pca[[1]],pca[[5]])
plotRGB(H_PC1_PC5,stretch='lin')
```



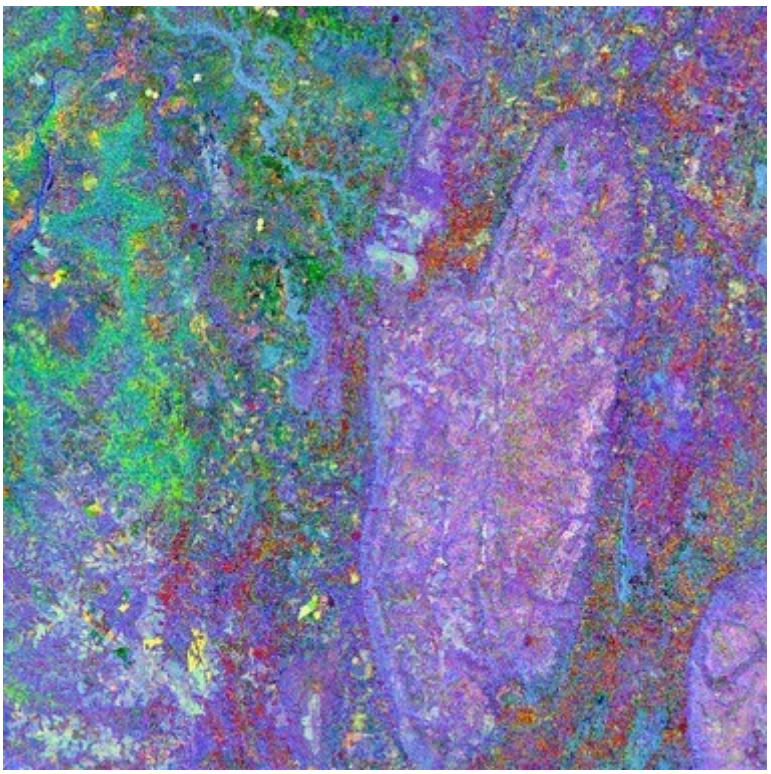
H, PC5, PC6:

```
> H_PC5_PC6<-stack(ihs[[2]],pca[[5]],pca[[6]])
> plotRGB(H_PC5_PC6,stretch='lin')
```



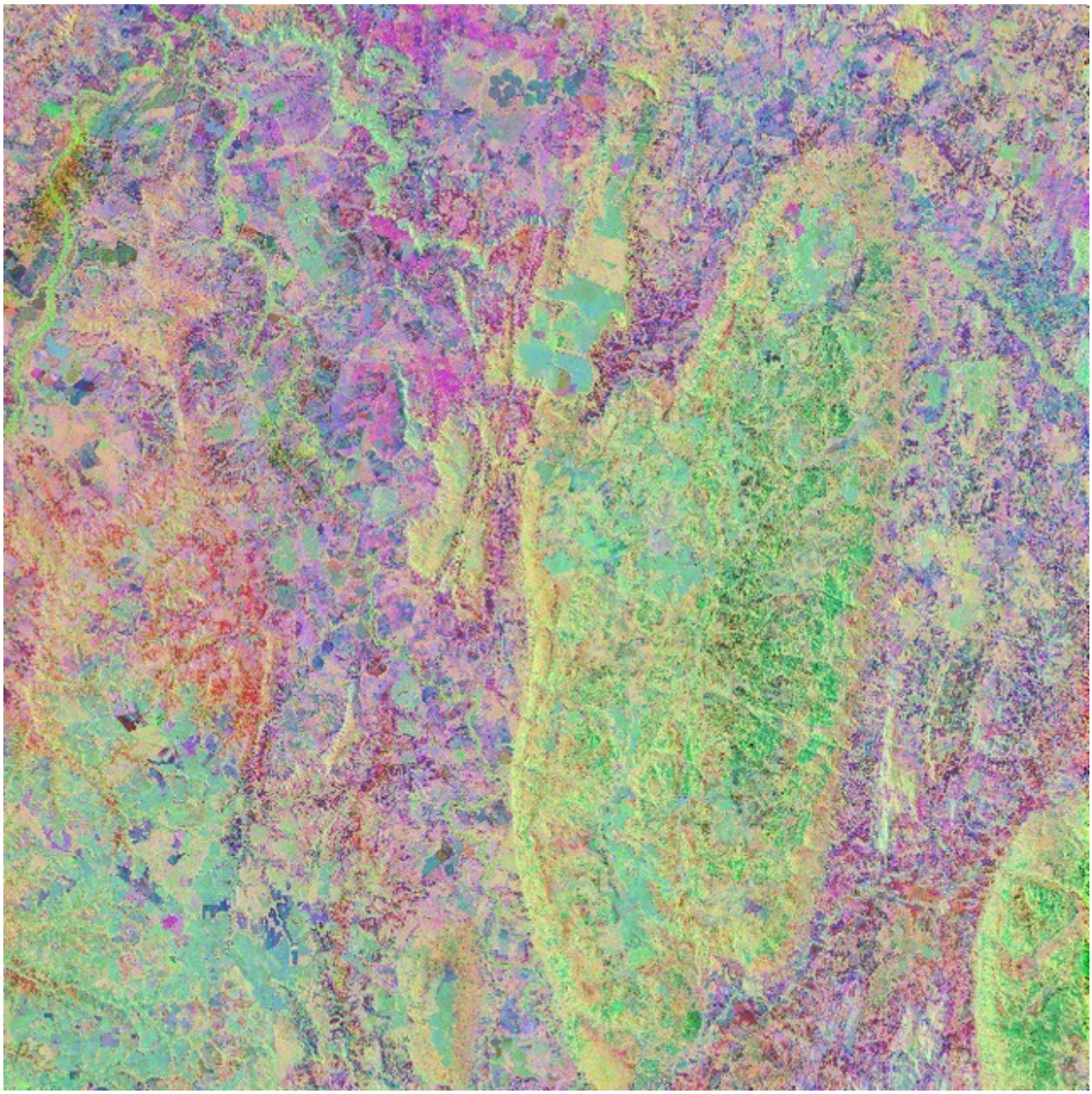
PC6_PC7_PC1:

```
> PC6_PC7_PC1<-stack(pca[[6]],pca[[7]],pca[[1]])
> plotRGB(PC6_PC7_PC1,stretch='lin')
```



Adicionando sombra de relevo ao H_PC1_PC5:

```
> relevo<-raster('hillshade.tif')
> plotRGB(H_PC1_PC5,stretch='lin')
> plot(relevo,add=TRUE,legend=FALSE,axes=FALSE,col=grey(0:100/100),alpha=0.35)
```



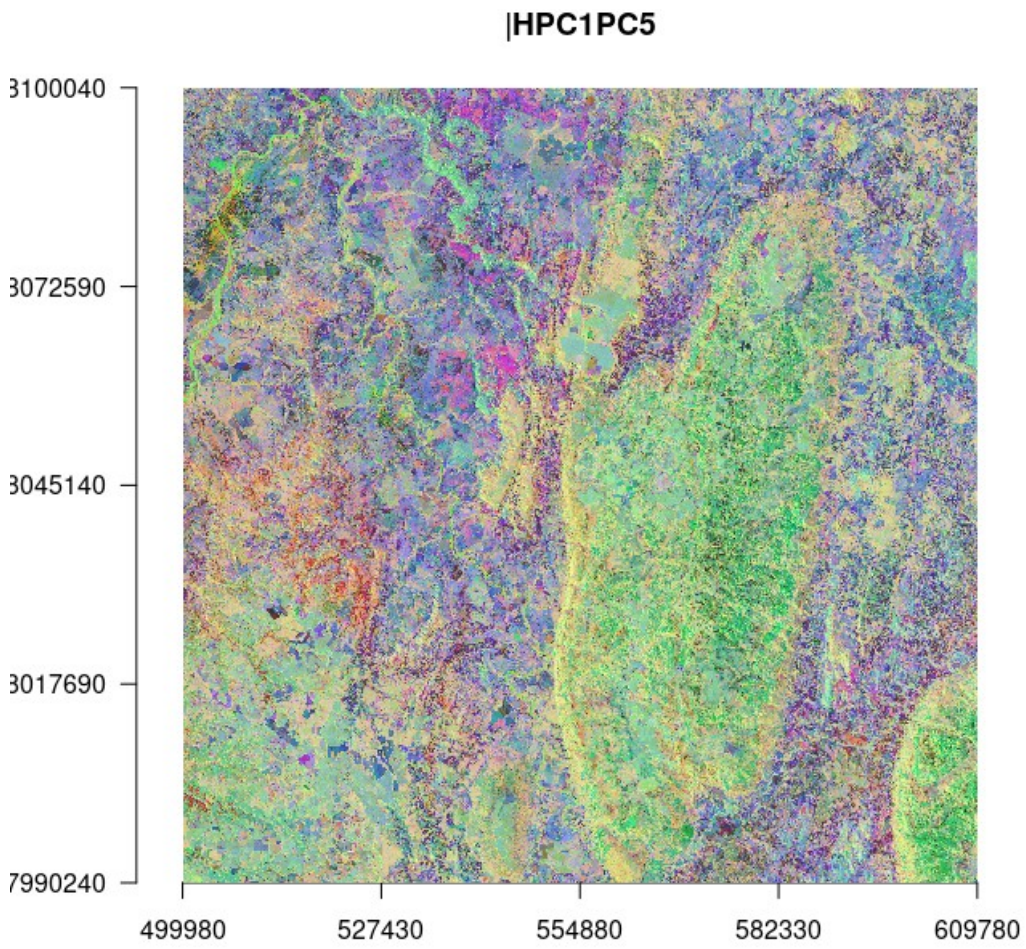
Agora, para finalizar a parte relacionada a processamento de imagem, vamos executar uma classificação não supervisionada pelo método K-means.

Vamos executar a classificação usand a composição H, PCI1 e PCI5 que apresentou melhor resultado em separar a litologia.

```
> library('raster')
> library('cluster')
> library('randomForest')
> relevo<-raster('hillshade.tif')
> img<-stack('H.tiff','pci1.tif','pci5.tif')
```

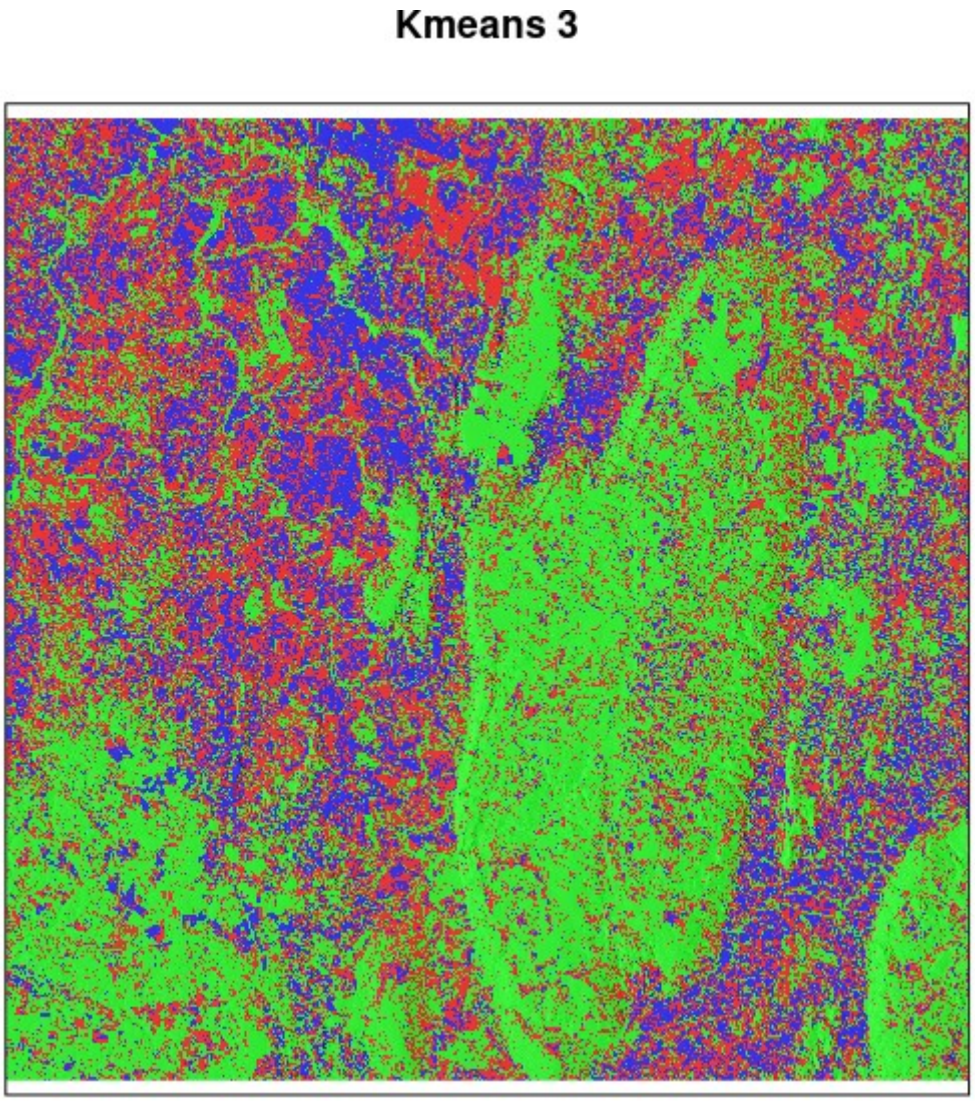
Com o código abaixo iremos criar cinco classificações k-means com 3, 6, 9, 12 e 15 classes. Esse cálculo poderá demorar até duas horas para ser concluído uma vez que estamos aplicando em uma cena completa do Sentinel2.

```
> ar <- getValues(img) #carregando o raster em um vetor
> values <- which(!is.na(ar)) #índice de valores válidos (valores não NA)
> ar <- na.omit(ar) #removendo NA do vetor
> Result <- clara(ar,3,samples=500,metric="manhattan",pamLike=T)
> #criando raster a ser classificado
> kmraster3 <- raster(img)
> kmraster3[values] <- Result$clustering #carregando valores da classe
> Result <- clara(ar,6,samples=500,metric="manhattan",pamLike=T)
> kmraster6 <- raster(img)
> kmraster6[values] <- Result$clustering
> Result <- clara(ar,9,samples=500,metric="manhattan",pamLike=T)
> kmraster9 <- raster(img)
> kmraster9[values] <- Result$clustering
> Result <- clara(ar,12,samples=500,metric="manhattan",pamLike=T)
> kmraster12 <- raster(img)
> kmraster12[values] <- Result$clustering
> Result <- clara(ar,15,samples=500,metric="manhattan",pamLike=T)
> kmraster15 <- raster(img)
> kmraster15[values] <- Result$clustering
> par(mfrow=c(2,3))
> plotRGB(img,stretch='lin',axes=T,main='HPC1PC5')
> plot(relevo,add=TRUE,legend=F,axes=F,col=grey(0:100/100),alpha=0.3)
```



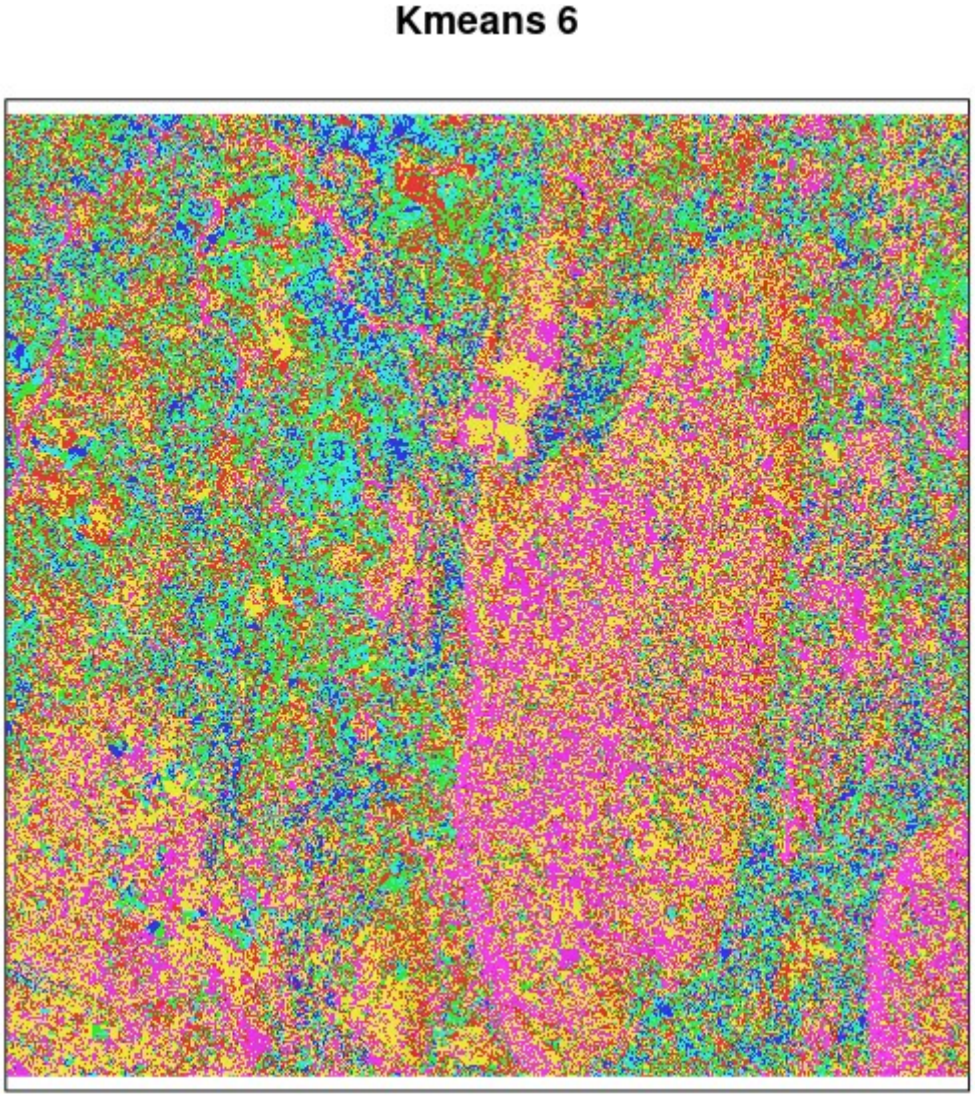
```
> plot(kmraster3, legend=F,axes=F, col = rainbow(3),main='Kmeans 3')
> plot(relevo,add=TRUE,legend=F,axes=F,col=grey(0:100/100),alpha=0.3)
```

O resultado com 3 classes foi satisfatório em separar categorias principais.



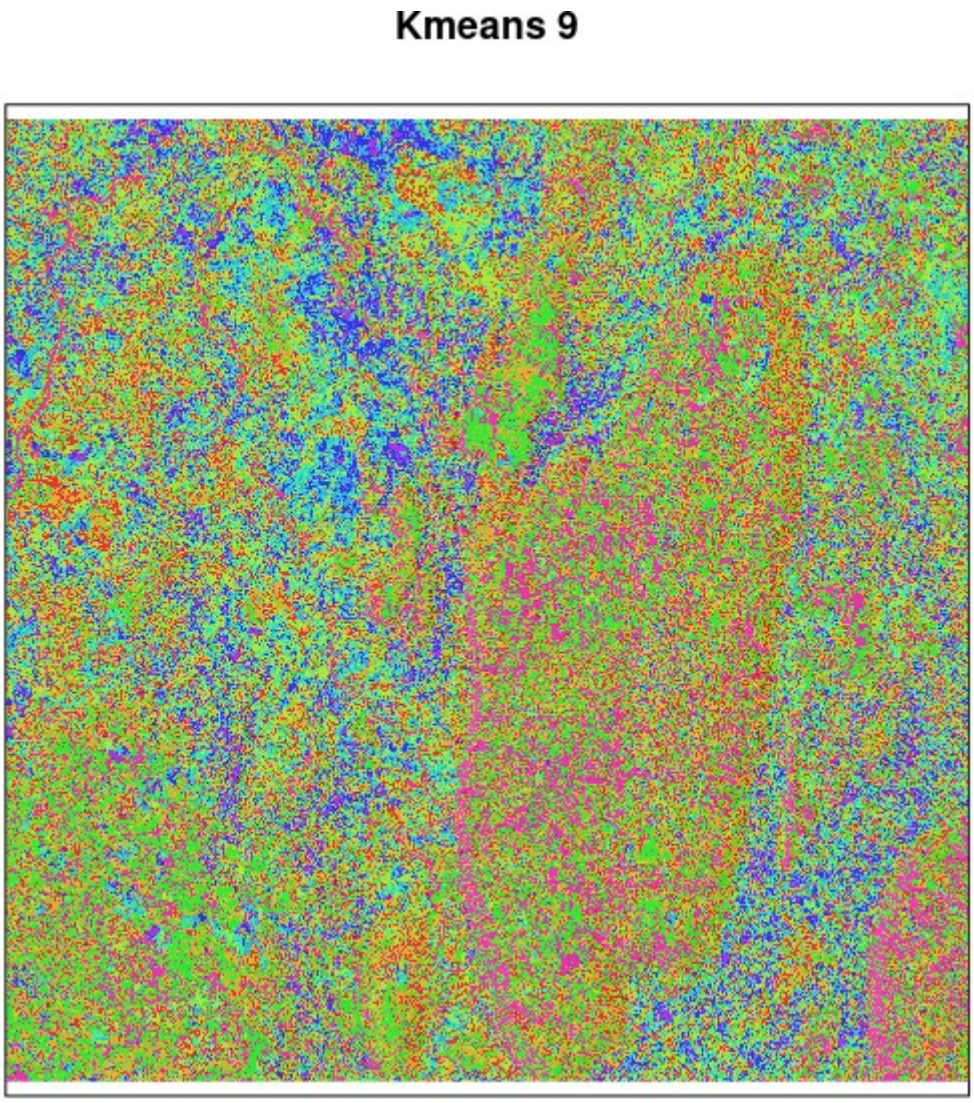
```
> plot(kmraster6, legend=F,axes=F, col = rainbow(6),main='Kmeans 6')
> plot(relevo,add=TRUE,legend=F,axes=F,col=grey(0:100/100),alpha=0.3)
```

O resultado com 6 classes inicia a separação mais refinada mas não o suficiente para dar uma resolução melhor.



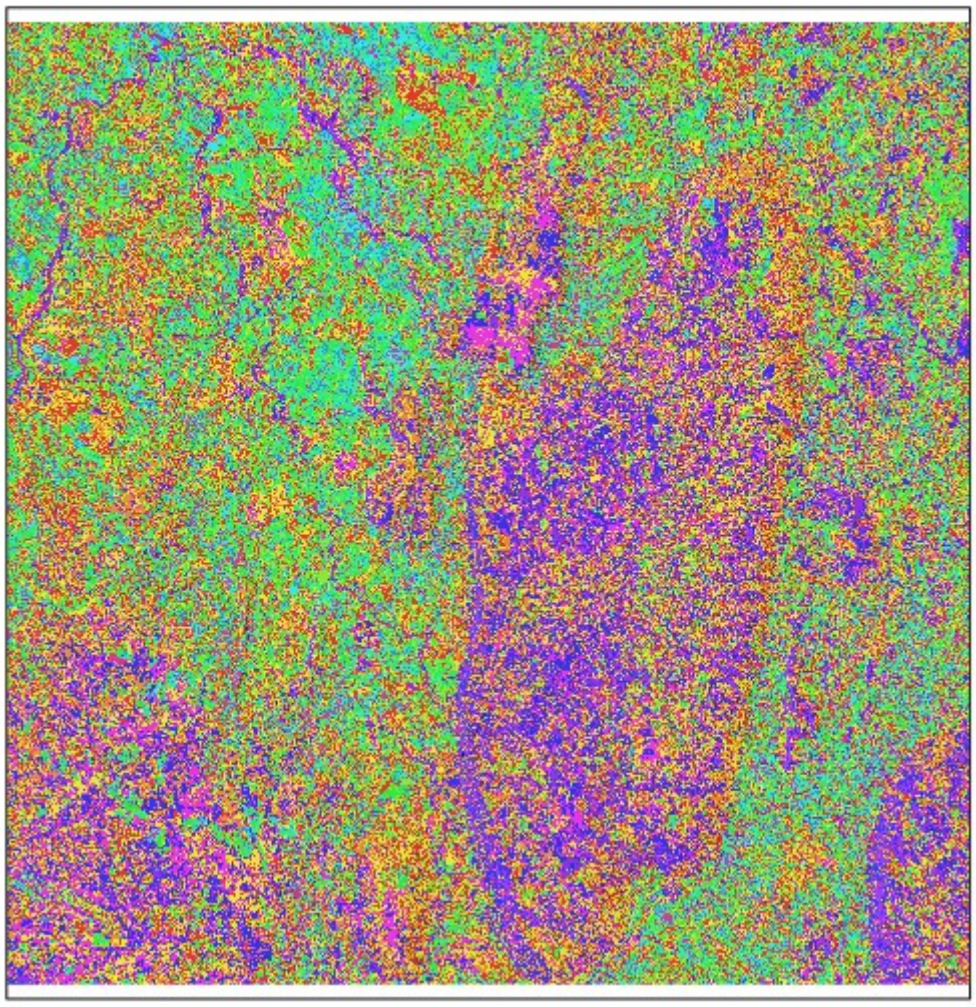
```
> plot(kmraster9, legend=F,axes=F, col = rainbow(9),main='Kmeans 9')
> plot(relevo,add=TRUE,legend=F,axes=F,col=grey(0:100/100),alpha=0.3)
```

Com 9 classes a resolução refinada melhora e se inicia a separação dos litotipos principais.



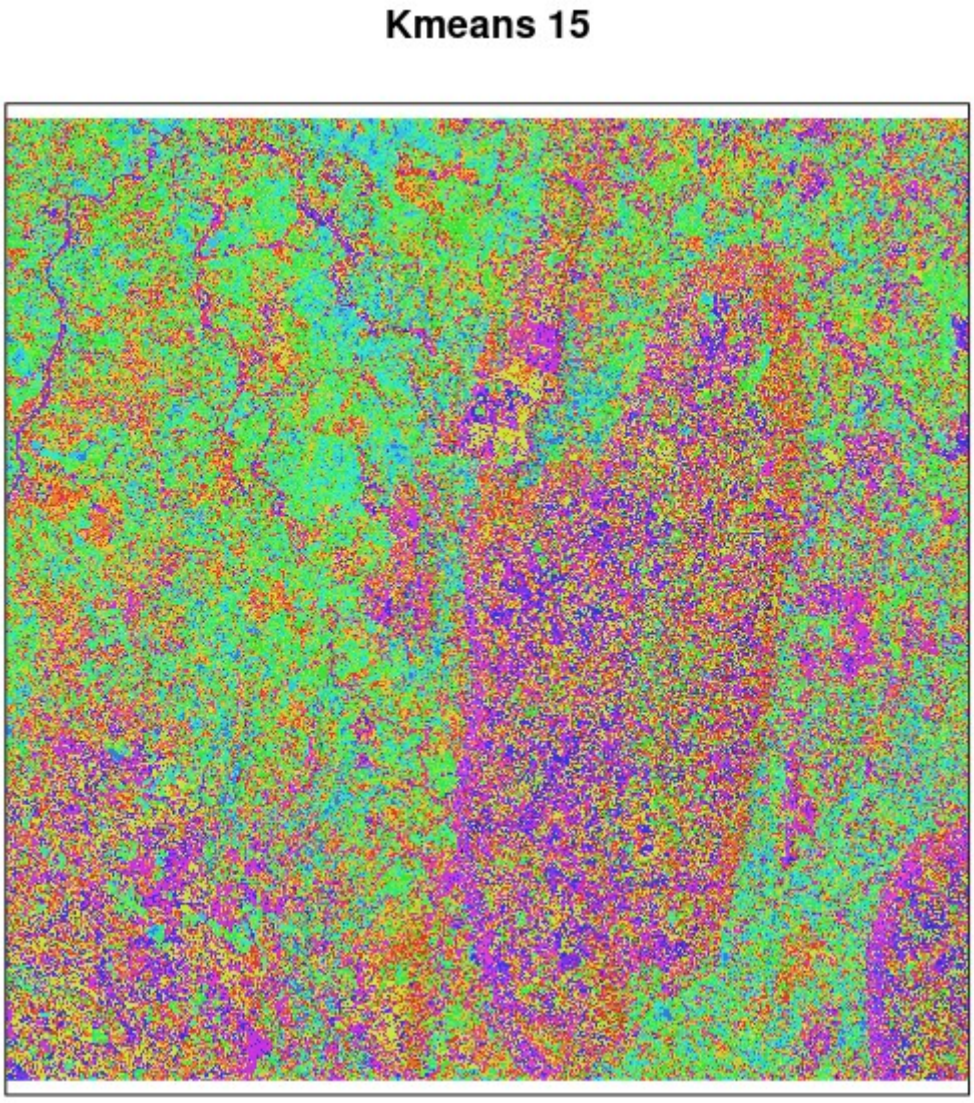
```
> plot(kmraster12, legend=F,axes=F, col = rainbow(12),main='Kmeans 12')
> plot(relevo,add=TRUE,legend=F,axes=F,col=grey(0:100/100),alpha=0.3)
```

Com 12 classes já é nítida a separação dos litotipos principais.



```
> plot(kmraster15, legend=F,axes=F, col = rainbow(15),main='Kmeans 15')
> plot(relevo,add=TRUE,legend=F,axes=F,col=grey(0:100/100),alpha=0.3)
```

Com 15 classes vemos uma boa separação dos litotipos semelhante às diferenciadas na imagem original.



3.3 – Dados geofísicos e outros dados espaciais (linhas, polígonos)

3.3.1 - Dados Magnetometria

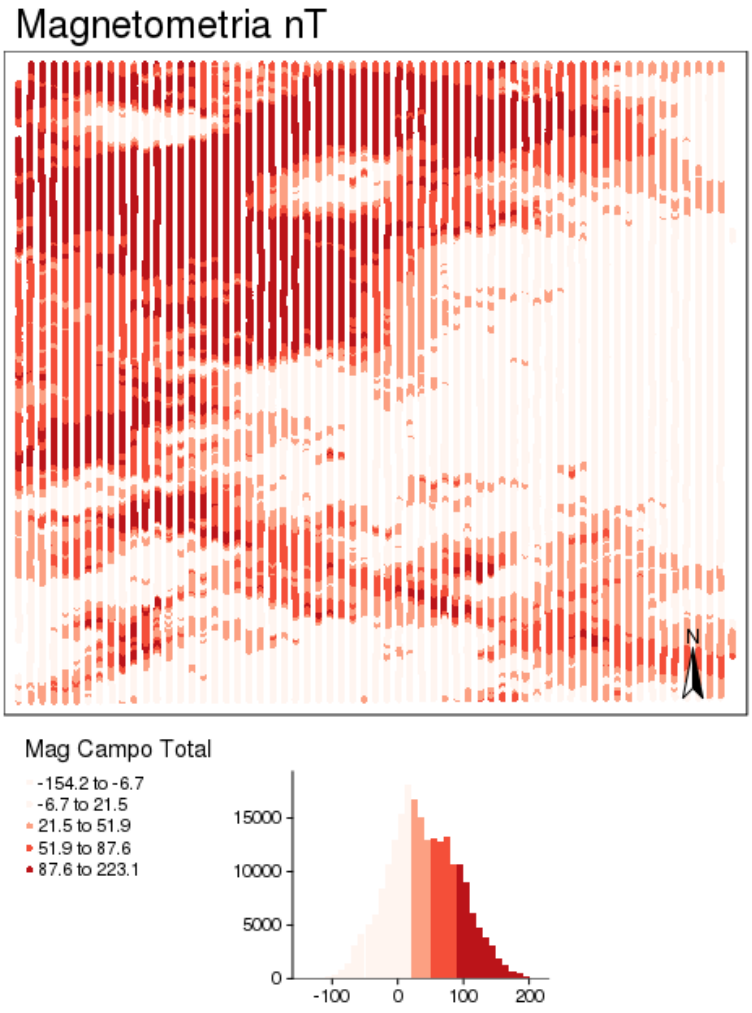
Vamos usar uma parte de um levantamento de magnetometria do CPRM feito no estado do Pará (Rio Iriri) como exemplo.

Vamos criar um SpatialPointsDataFrame a partir do arquivo csv. O datum é WGS-84 em UTM Zona 22 S (EPSG 32722):

```
> library(raster)
> mag<-read.csv('mag1131.csv',stringsAsFactors=FALSE)
> coordinates(mag)<-~X+Y
> crs(mag)<-CRS('+init=epsg:32722')
```

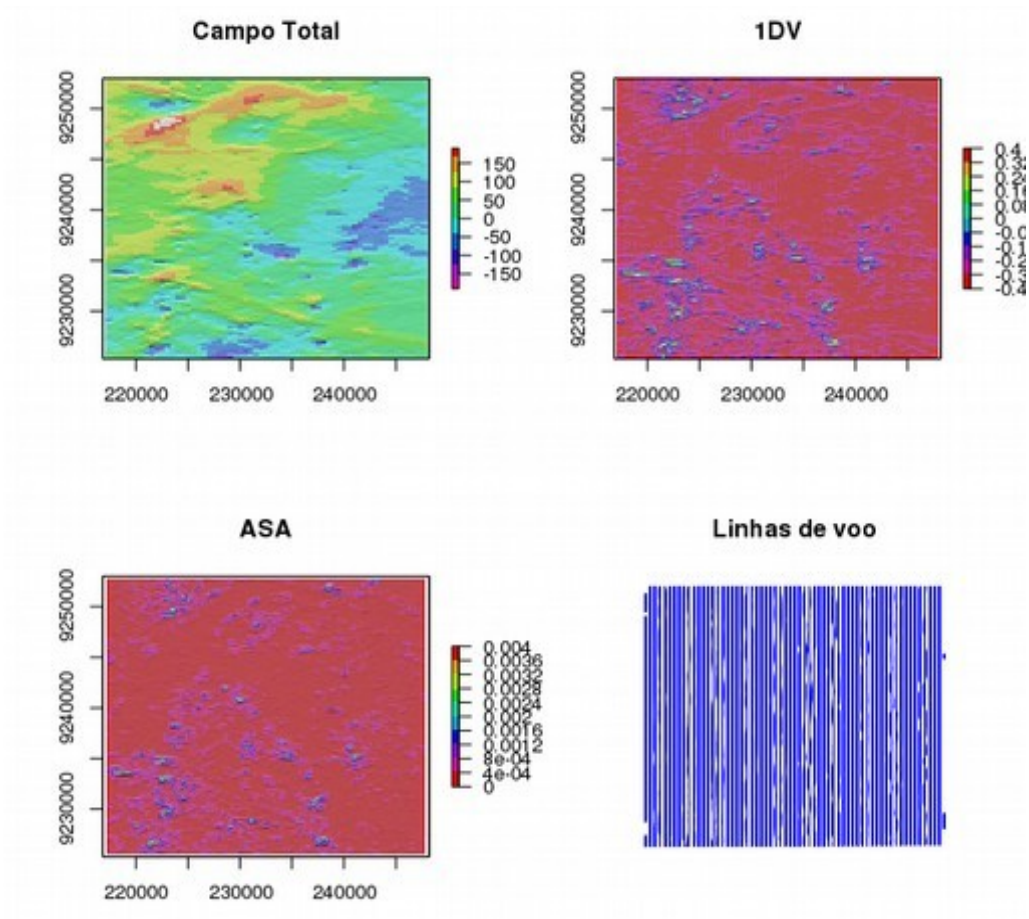
Visualizado mapa dos pontos e valores de MAGIGRF:

```
> library(tmap)
> tm_shape(mag) + tm_dots(size=0.08,col = "MAGIGRF", palette = "Reds", style =
"quantile", title='Mag Campo Total',legend.hist=TRUE)+
tm_layout(main.title='Magnetometria nT', legend.outside.position = c("bottom"),
legend.outside=TRUE,legend.stack ='horizontal',legend.hist.height=0.75,
legend.hist.width=0.95)+tm_compass()
```



Vamos agora interpolar os dados e criar uma imagem raster do campo total usando o método vizinho mais próximo e vamos extrair a primeira e segunda derivadas do campo total:

```
> library(gstat)
> rast<-raster(mag,res=100)
> metodo<-gstat(formula=MAGIGRF~1,location=mag,nmax=5,set=list(idp=1.5))
> viz<-interpolate(rast,metodo)
> gradiente<-terrain(viz,opt='slope',unit='tangent')
> aspecto<-terrain(viz,opt='aspect',unit='radians')
> gradiente.cor<-gradiente-0.38
> gradiente2<-terrain(gradiente.cor,opt='slope',unit='tangent')
> aspecto2<-terrain(gradiente.cor,opt='aspect',unit='radians')
> gradiente3<-terrain(gradiente2,opt='slope',unit='tangent')
> aspecto3<-terrain(gradiente2,opt='aspect',unit='radians')
> relevo<-hillShade(gradiente,aspecto,angle=45,direction=0)
> relevo2<-hillShade(gradiente2,aspecto2,angle=45,direction=0)
> relevo3<-hillShade(gradiente3,aspecto3,angle=45,direction=0)
> par(mfrow=c(2,2))
> plot(viz,col=rev(rainbow(10)),zlim=c(-190,190),main='Campo Total')
> plot(relevo,add=TRUE,alpha=0.55,col=grey(0:100/100),legend=F)
> plot(gradiente.cor,col=rev(rainbow(10,start=0,end=.99)),lab.breaks=seq(-
0.4,0.4,0.08),breaks=seq(-0.4,0.4,0.08),zlim=c(-0.38,0.3),main='1DV')
> plot(relevo2,add=TRUE,alpha=0.55,col=grey(0:100/100),legend=F)
> plot(gradiente2,col=rev(rainbow(10,start=0,end=0.99)),lab.breaks=seq(0,
0.004,0.0004),breaks=seq(0,0.004,0.0004),zlim=c(0,0.004),main='ASA')
> plot(relevo3,add=TRUE,alpha=0.55,col=grey(0:100/100),legend=F)
> plot(mag,cex=0.0001,col='blue',main='Linhas de voo')
```



3.3.2 - Dados Gamaespectrometria

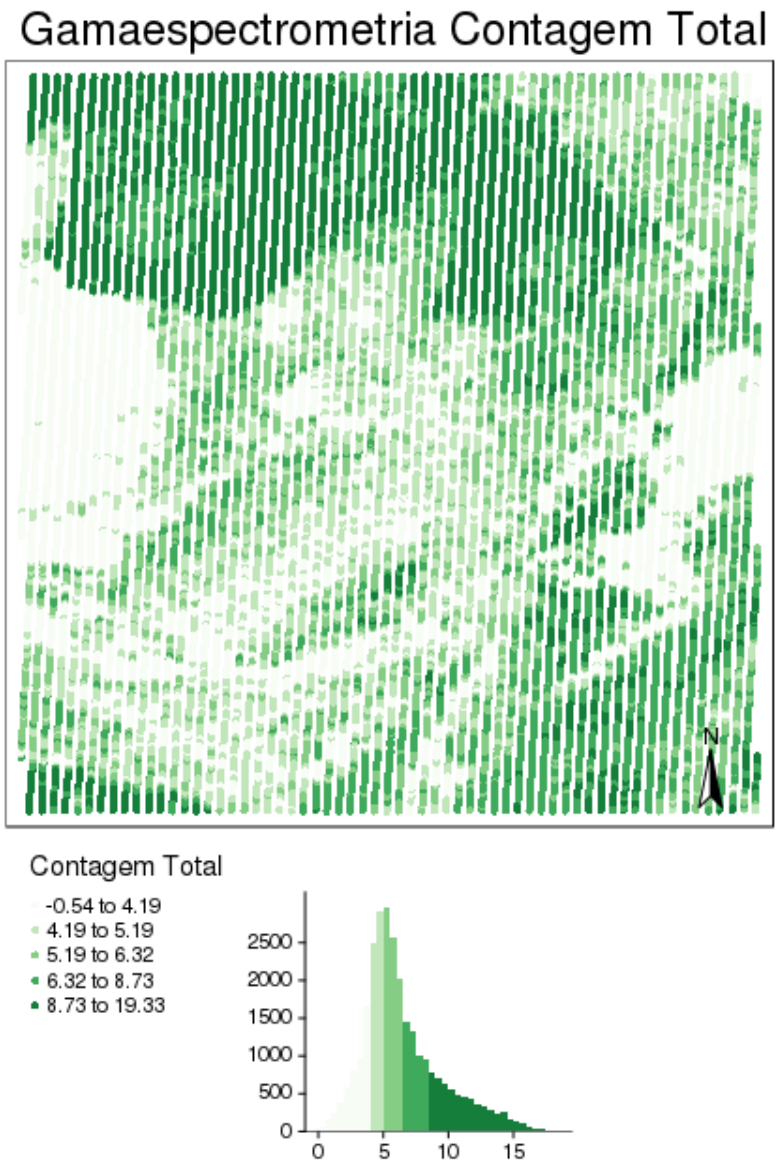
Vamos usar uma parte de um levantamento de gamaespectrometria do CPRM feito no estado do Rio Grande do Sul como exemplo.

Vamos criar um SpatialPointsDataFrame a partir do arquivo csv. O datum é WGS-84 em UTM Zona 21 S (EPSG 32721):

```
> library(raster)
> gama<-read.csv('gama1100.csv',stringsAsFactors=FALSE)
> coordinates(gama)<-~X+Y
> crs(gama)<-CRS('+init=epsg:32721')
```

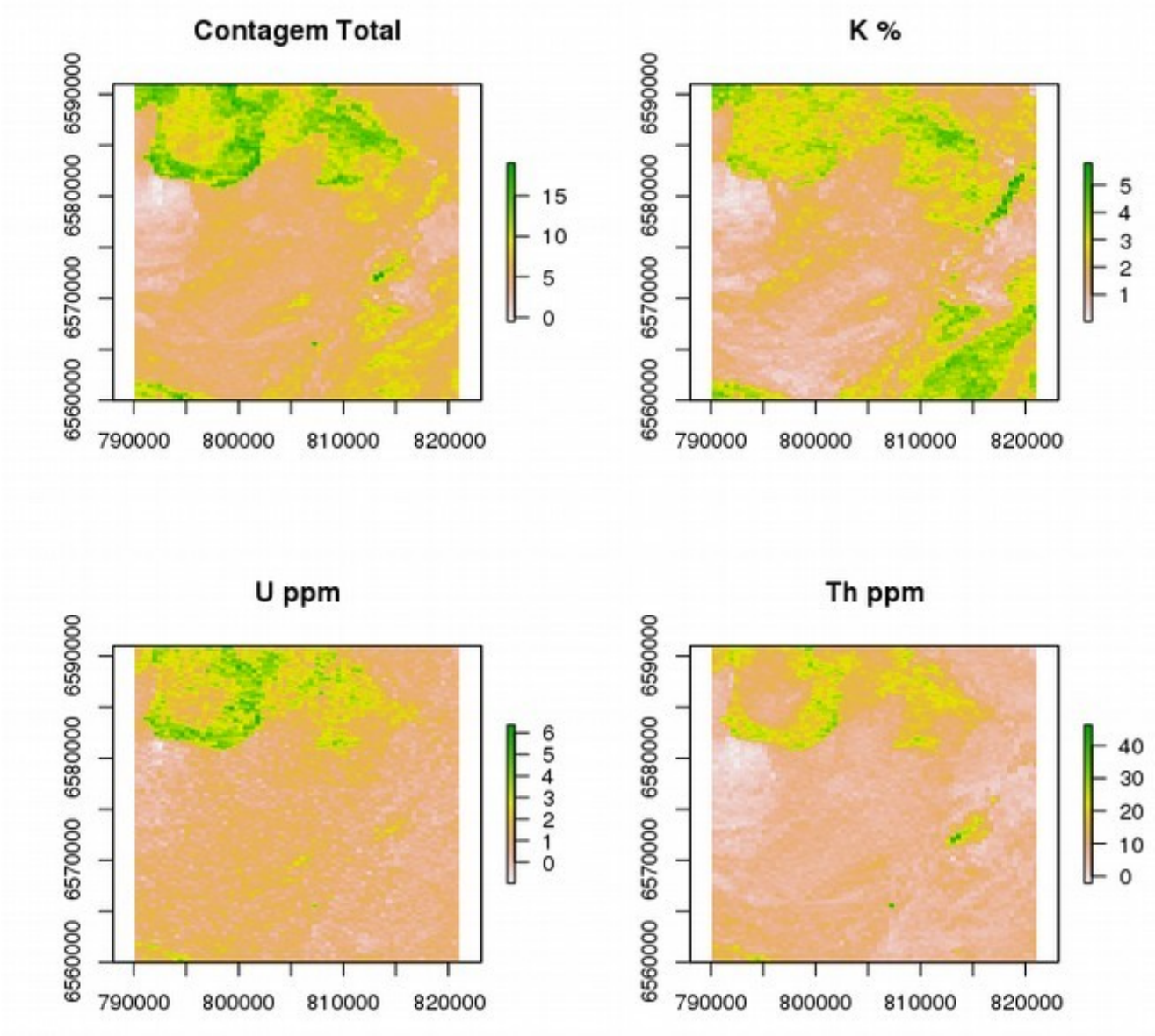
Visualizado mapa dos pontos e valores de CTEXP:

```
> library(tmap)
> tm_shape(gama) + tm_dots(size=0.08,col = "CTEXP", palette = "Greens", style =
"quantile", title='Contagem Total',legend.hist=TRUE)+
tm_layout(main.title='Gamaespectrometria Contagem Total',
legend.outside.position = c("bottom"), legend.outside=TRUE,legend.stack
='horizontal',legend.hist.height=0.75, legend.hist.width=0.95)+tm_compass()
```



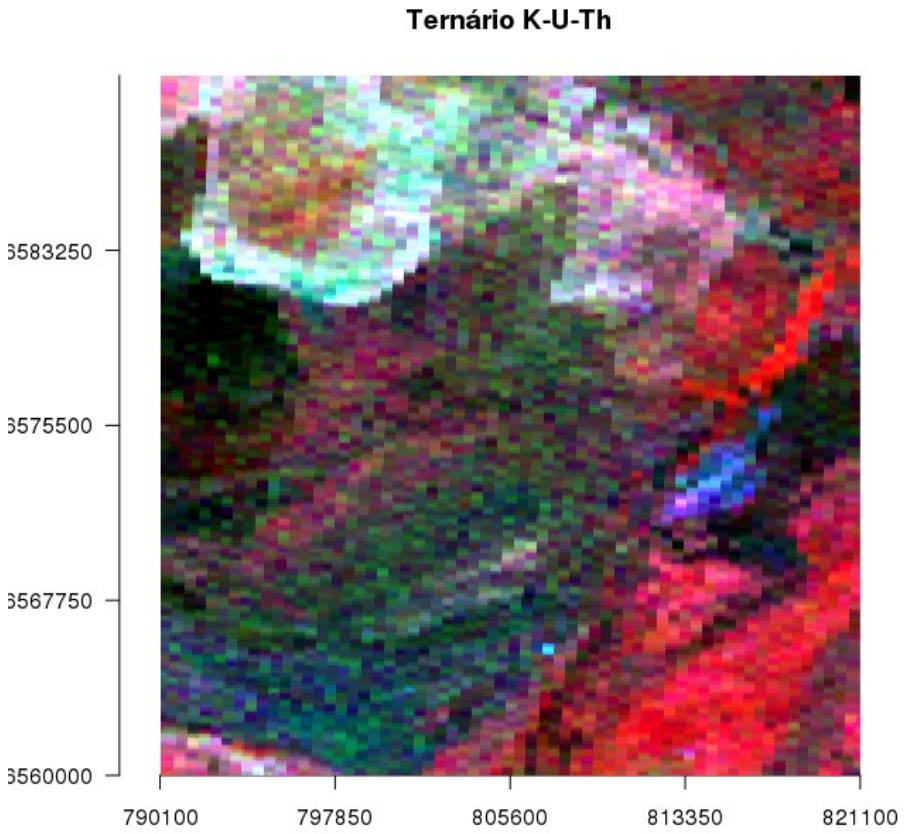
Interpolando rasters com os dados de Contagem Total, U, Th e K:

```
> library(gstat)
> rast<-raster(gama,res=100)
> metodo<-gstat(formula=CTEXP~1,location=gama,nmax=5,set=list(idp=1.5))
> ct<-interpolate(rast,metodo)
> metodo<-gstat(formula=Kperc~1,location=gama,nmax=5,set=list(idp=1.5))
> K<-interpolate(rast,metodo)
> metodo<-gstat(formula=eU~1,location=gama,nmax=5,set=list(idp=1.5))
> U<-interpolate(rast,metodo)
> metodo<-gstat(formula=eTh~1,location=gama,nmax=5,set=list(idp=1.5))
> Th<-interpolate(rast,metodo)
> par(mfrow=c(2,2))
> plot(ct,main='Contagem Total')
> plot(K,main='K %')
> plot(U,main='U ppm')
> plot(Th,main='Th ppm'
```



Gerando um mapa ternário RGB com os rasters K, U e Th:

```
> KUTh←stack(K,U,Th)
> dev.off()
> plotRGB(KUTh,stretch='lin',main='Ternário K-U-Th',axes=TRUE)
```



3.3.3 - Dados de Perfilagem de furo

Usaremos um arquivo LAS (Log ASCII Standard) muito usado para em perfilagem geofísica em furos de exploração de petróleo e mineral.

Lendo arquivo LAS e criando data.frame com os dados:

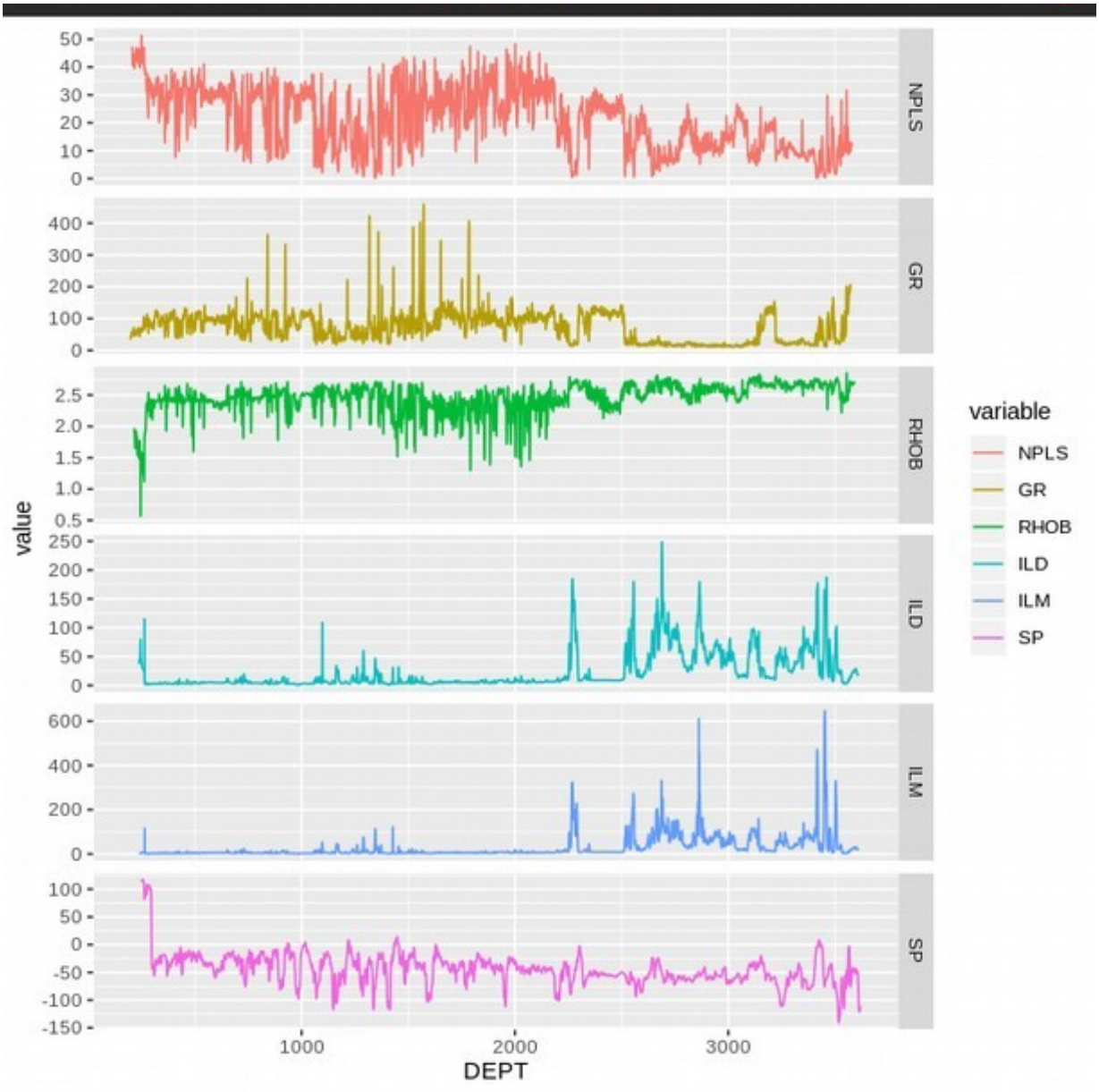
```
> las<-readLines('1046531020.las')
> data<-''
> hd<-''
> a<-1
> go<-0
> for (i in 1:length(las)){
+ if(substr(las[i], 1, 2) == "~A"){
+ data<- las[(i+1):length(las)]
+ break
+ }
}
> las<-readLines('1046531020.las')
> for (i in 1:length(las)){
+ if(substr(las[i], 1, 2) == "~C"){go<-1}
+ if(substr(las[i+1], 1, 2) == "~P" || substr(las[i+1], 1, 2) == "~A"
+ || substr(las[i+1], 1, 2) == "~O"){break}
+ if(go==1){value<-strsplit(trimws(las[i+1],"l"),'\\s{1,}')[[1]]
+ hd[a]<-value[1]
+ a<-a+1
+ }
}
```

```
> las.data<-read.table(header=TRUE, text=data)
> names(las.data)<-hd
> las.data[las.data==-999.2500]<-NA
> head(las.data,n=2)
   DEPT CILD CALN CALD GR ILD ILM MINV LWTLB CALM MNOR NPLS DPLS RXRT SFL SP
1 194.5   NA   NA   NA NA  NA  NA   NA    NA   NA   NA   NA   NA   NA  NA NA
2 195.0   NA   NA   NA NA  NA  NA   NA    NA   NA   NA   NA   NA   NA  NA NA
  DRHO RHOB PE INV_RF NOR_RF
1   NA   NA NA     NA     NA
2   NA   NA NA     NA     NA
```

Este processo foi feito para LAS versão 2.0 (não envelopados), pequena adaptação será necessária para dados envelopados.

Selecionando e plotando algumas curvas:

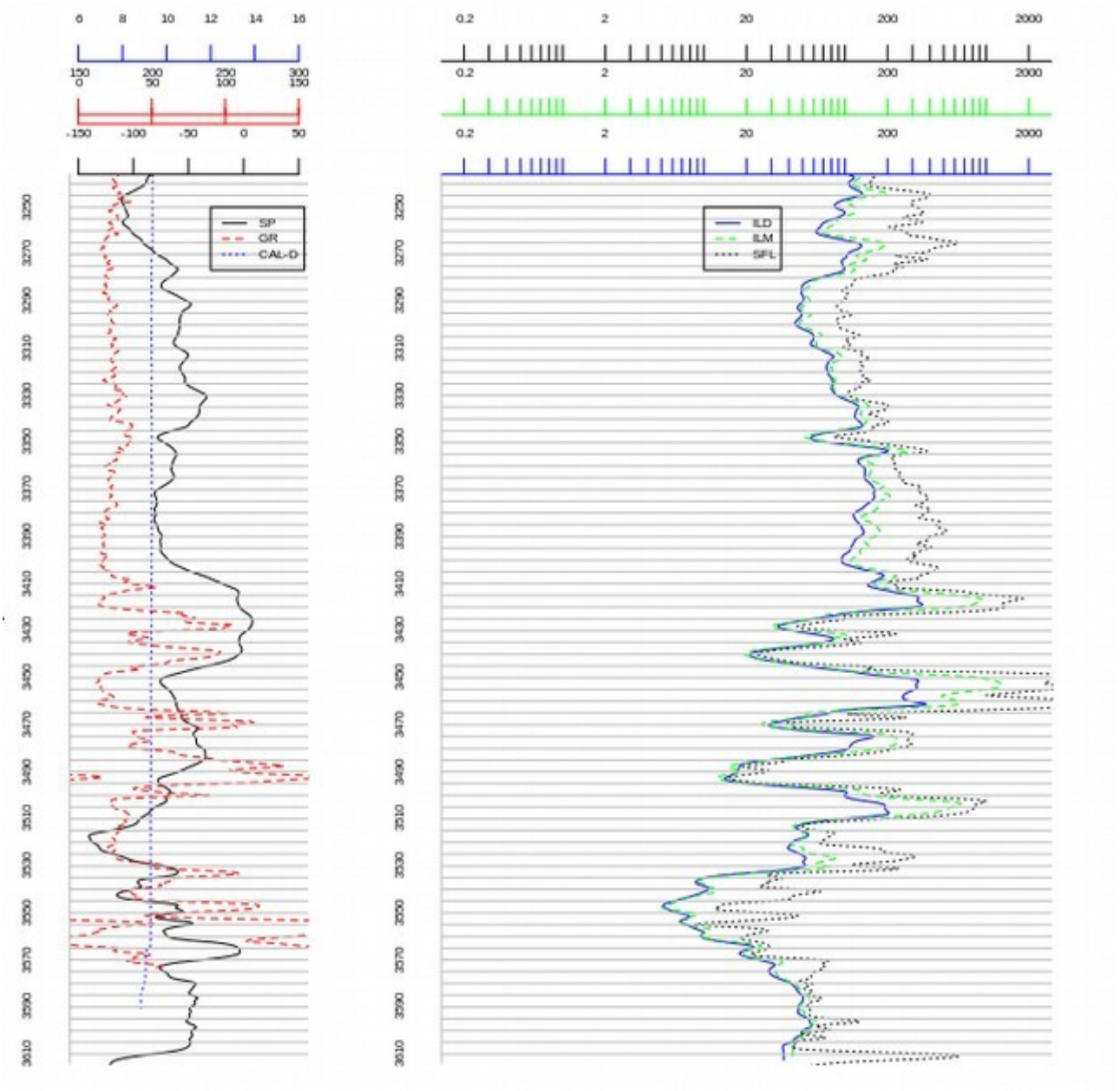
```
> library(reshape2)
> library(ggplot2)
> mm<-melt(las.data,measure.vars=c('NPLS','GR','RHOB','ILD','ILM','SP'))
> ggplot(mm, aes(x = DEPT, y = value)) +geom_line(aes(color = variable))
+facet_grid(variable ~ ., scales = "free_y")
```



Plotando o LAS no formato tradicional:

```
> par(mar=c(1,2,5.5,2) + 0.1)
> layout(matrix(c(1,2),nrow=1), widths=c(1,2))
> plot(las.data$SP,las.data$DEPT,axes=FALSE,xlim=c(-150,50),type='l',xlab='',
+ ylab='',col='black',ylim=c(3600,3250))
> axis(3, xlim=c(-150,50),col='black',lwd=1,cex.lab=1,cex.axis=0.5)
> par(new=TRUE)
> plot(las.data$GR,las.data$DEPT,  axes=FALSE, xlim=c(0,150), type='l',xlab='',
+ ylab='',col='red',lty=2, ylim=c(3600,3250),lwd=1)
> axis(3, xlim=c(0,150),col='red',lwd=1,line=1.6,cex.lab=1,cex.axis=0.5)
> par(new=TRUE)
> plot(las.data$GR,las.data$DEPT,  axes=FALSE, xlim=c(150,300),
+ type='l',xlab='', ylab='',col='red', lty=2, ylim=c(3600,3250),lwd=1)
> axis(3, xlim=c(0,150),col='red',lwd=1,line=1.9,cex.lab=1,cex.axis=0.5)
> par(new=TRUE)
> plot(las.data$CALD,las.data$DEPT,  axes=FALSE, xlim=c(6,16),
+ type='l',xlab='', ylab='',col='blue', lty=3, ylim=c(3600,3250),lwd=1)
> axis(3, xlim=c(6,16),col='blue',lwd=1,line=3.6,cex.lab=1,cex.axis=0.5)
> axis(2,pretty(range(las.data$DEPT),500),cex.lab=1,tck=1,cex.axis=0.5,col=
+ 'gray')
> mtext('depth',side=2,col="black",line=2)
> legend(x=12,y=3250,legend=c('SP','GR','CAL-D'),lty=c(1,2,3),col=c('black',
+ 'red','blue'),cex=0.5)
> plot(las.data$ILD,las.data$DEPT,axes=FALSE,xlim=c(10^-1,10^3),type='l',
+ xlab='', ylab='',col='blue',ylim=c(3600,3250),log='x')
> at.y <- outer(1:9, (.5*10^(-1:3)))
> lab.y <- ifelse(log10(at.y) %% 1 == 0, at.y*2, NA)
> axis(3, xlim=c(10^-1,10^3),col='blue',lwd=1,cex.lab=1,cex.axis=0.5, at=at.y,
+ labels=lab.y, las=1)
> par(new=TRUE)
> plot(las.data$ILM,las.data$DEPT,  axes=FALSE, xlim=c(10^-1,10^3),
+ type='l',xlab='', ylab='',col='green',lty=2, ylim=c(3600,3250),lwd=1,log='x')
> axis(3, xlim=c(10^-1,10^3),col='green',lwd=1,line=1.9,cex.lab=1,cex.axis=0.5,
+ at=at.y,labels=lab.y, las=1)
> par(new=TRUE)
> plot(las.data$SFL,las.data$DEPT,  axes=FALSE, xlim=c(10^-1,10^3),
+ type='l',xlab='', ylab='',col='black',lty=3, ylim=c(3600,3250),lwd=1,log='x')
> axis(3, xlim=c(10^-1,10^3),col='black',lwd=1,line=3.6,cex.lab=1,cex.axis=0.5,
+ at=at.y,labels=lab.y, las=1)
> axis(2,pretty(range(las.data$DEPT),500),cex.lab=1,tck=1,cex.axis=0.5,col=
+ 'gray')
> legend(x=5,y=3250,legend=c('ILD','ILM','SFL'),lty=c(1,2,3),col=c('blue',
+ 'green','black'),cex=0.5)
```

O resultado será:



No formato data.frame os dados de um arquivo LAS podem ser facilmente usados para caracterização de reservatórios, predição de litologia, e outros tratamentos.

3.3.4 - Dados sísmico e GPR em arquivo seg-Y

Usaremos um exemplo de como ler e interagir com informações oriundas de um arquivo de dados sísmico e GPR. Este formato é um pouquinho mais complexo e apresenta muitas variações de formato. O exemplo é ilustrativo e funciona com boa parte dos arquivos seg-Y.

Arquivos seg-Y são divididos em três partes principais (**genericamente falando**); Cabeçalho geral do arquivo (3200 bytes do cabeçalho texto mais 400 bytes do cabeçalho binário), cabeçalho binário do traço (240 bytes) e dado binário do traço.

Vamos então mostrar como carregar em R um arquivo seg-Y por etapas. Primeiro preparando os vetores de nomes de cabeçalho e o conector do arquivo de dados e lendo o cabeçalho texto do arquivo:

```
> hdbn<-c('JOB_ID_NO','LINE_NUMBER','REEL_NUMBER','TRACES_PER_RECORD',
'AUXS_PER_RECORD','SAMPLE_RATE','SAMPLE_RATE_FIELD','NSAMPLES','NSAMPLES_FIELD'
,'FORMAT_CODE','ENSEMBLE_FOLD','TRACE_SORT','VERTICAL_SUM','SWEEP_FREQ_START','
SWEEP_FREQ_END','SWEEP_FREQ_LENGTH','SWEEP_TYPE','SWEEP_TRACE_NO','SWEEP_TAPER_
LENGTH_START','SWEEP_TAPER_LENGTH_END','SWEEP_TAPER_TYPE','CORRELATED_TRACES','
BINARY_GAIN','AMP_RECOVERY_METHOD','UNITS','SIGNAL_POLARITY','VIBRATOR_POL_CODE
','UNUSED','SEGY_FORMAT_REVISION_NO','SEGY_FIXEDLEN_FLAG','SEGY_NO_TEXTFHEADERS
','UNUSED')
> con <- file("1104-30A.segy","rb")
> library(float)
> cabe.texto<-c('bogus','data')
> for(i in 1:40){
+  dec<-readBin(con,"raw",size=1,n=80)
+  cabe.texto[i]<-paste0(iconv(rawToChar(dec,multiple=T),'cp500','utf8'),
collapse='')
+ }
> cabe.texto
 [1] "C 1 CLIENT MMS                    COMPANY GECO                  CREW NO "
 [2] "C 2 LINE 1104-30A   AREA PHASE-30A             MAP ID                    "
 [3] "C 3 REEL NO 17784MIG0 DAY-START OF REEL     YEAR      OBSERVER           "
 [4] "C 4 INSTRUMENT: MFG           MODEL            SERIAL NO                 "
 [5] "C 5 DATA TRACES/RECORD        AUXILIARY TRACES/RECORD        CDP FOLD   "
 [6] "C 6 SAMPLE INTERVAL  4 SAMPLES/TRACE  2750 BITS/IN    0 BYTES/SAMPL   0"
 [7] "C 7 RECORDING FORMAT        FORMAT THIS REEL        MEASUREMENT SYSTEM  "
 [8] "C 8 SAMPLE CODE: FLOATING PT 032 FIXED PT     FIXED PT-GAIN     CORRELAT"
 [9] "C 9 GAIN  TYPE: FIXED     BINARY     FLOATING POINT     OTHER           "
[10] "C10 FILTERS: ALIAS     HZ  NOTCH     HZ  BAND     -     HZ  SLOPE    -  "
[11] "C11 SOURCE: TYPE AIRGUN    NUMBER/POINT         POINT INTERVAL           "
[12] "C12     PATTERN:                           LENGTH        WIDTH           "
[13] "C13 SWEEP: START     HZ  END     HZ  LENGTH      MS  CHANNEL NO     TYPE"
[14] "C14 TAPER: START LENGTH       MS  END LENGTH       MS  TYPE              "
[15] "C15 SPREAD: OFFSET        MAX DISTANCE        GROUP INTERVAL            "
[16] "C16 GEOPHONES: PER GROUP     SPACING     FREQUENCY     MFG          MODE"
[17] "C17     PATTERN:                           LENGTH        WIDTH           "
[18] "C18 TRACES SORTED BY: RECORD     CDP     OTHER                           "
[19] "C19 AMPLITUDE RECOVERY: NONE      SPHERICAL DIV       AGC    OTHER      "
[20] "C20 PROJECTION : SPHEROID :                  "
[21] "C21 FAMILY 1 DEFAULTS: DATA TRACES/RECORD        AUXILIARY TRACES/RECORD"
[22] "C22                    SAMPLE INTERVAL         SAMPLES/TRACE             "
[23] "C23 FAMILY 3 DEFAULTS: SAMPLE INTERVAL         SAMPLES/TRACE             "
[24] "C24 CENTRAL MERIDIAN :  0  0 0.0 E    ORIGIN PARALLEL :  0  0 0.0 N     "
[25] "C25 FALSE NORTHING :  0.0 FALSE EASTING :  0.0 UNIT TO METER :0.000000  "
[26] ""
[27] "C27 MXYMRGOV 15.06.22 12- 8-88 A6602IM4 0526     1104-30                 "
[28] "C28                                                                      "
[29] "C29                                                                      "
```

```
[30] "C30                                                                 "
[31] "C31                                                                 "
[32] "C32                                                                 "
[33] "C33                                                                 "
[34] "C34                                                                 "
[35] "C35                                                                 "
[36] "C36                                                                 "
[37] "C37                                                                 "
[38] "C38                                                                 "
[39] "C39                                                                 "
[40] "C40 END EBCDIC                                                      "
```

Agora vamos ler o cabeçalho binário do arquivo:

```
> seek(con,3200) #posicionando arquivo no byte 3201 (começa com 0)
> cabe.bin<-c(1.0,1.7)
> passo<-1
> for (i in passo:(passo+2)){
+ cabe.bin[passo]<-readBin(con,'int',size=4,endian='big')
+ passo<-passo+1
+ }
> for (i in passo:(passo+23)){
+ cabe.bin[passo]<-readBin(con,'int',size=2,endian='big')
+ passo<-passo+1
+ }
> cabe.bin[passo]<-0
> seek(con,3500)
[1] 3260
> passo<-passo+1
> for (i in passo:(passo+2)){
+ cabe.bin[passo]<-readBin(con,'int',size=2,endian='big')
+ passo<-passo+1
+ }
> cabe.bin[passo]<-0
> seek(con,3600)
[1] 3506
> cabecalho.bin<-data.frame(campo=hdbn,valor=cabe.bin)
> cabecalho.bin
                    campo valor
1                  JOB_ID_NO   526
2               LINE_NUMBER  1104
3               REEL_NUMBER    30
4         TRACES_PER_RECORD     1
5           AUXS_PER_RECORD     0
6               SAMPLE_RATE  4000
7         SAMPLE_RATE_FIELD     0
8                  NSAMPLES  2751
9            NSAMPLES_FIELD  2751
10              FORMAT_CODE     1
11             ENSEMBLE_FOLD     1
12                TRACE_SORT     4
13             VERTICAL_SUM     2
14         SWEEP_FREQ_START     0
15           SWEEP_FREQ_END     0
16        SWEEP_FREQ_LENGTH     0
17               SWEEP_TYPE     0
18           SWEEP_TRACE_NO     0
19 SWEEP_TAPER_LENGTH_START     0
20   SWEEP_TAPER_LENGTH_END     0
21         SWEEP_TAPER_TYPE     0
22        CORRELATED_TRACES     0
23              BINARY_GAIN     0
```

```
24      AMP_RECOVERY_METHOD     0
25                    UNITS     1
26          SIGNAL_POLARITY     0
27        VIBRATOR_POL_CODE     0
28                   UNUSED     0
29  SEGY_FORMAT_REVISION_NO     0
30       SEGY_FIXEDLEN_FLAG     0
31     SEGY_NO_TEXTFHEADERS     0
32                   UNUSED     0
```

Vamos agora criar o data.frame cabeçalho do traço e o vetor traço. Antes temos que calcular o número de traços do arquivo:

```
> tam.arq<-file.info('1104-30A.segy')$size
> tam.arq
[1] 50995140
> rep.cabecalho<-cabecalho.bin[31,2]
> rep.cabecalho
[1] 0
> numero.samples<-cabecalho.bin[8,2]
> numero.samples
[1] 2751
> num.tr<-(tam.arq-3600-3600*rep.cabecalho)/(240+numero.samples*4)
> num.tr
[1] 4535
```

Agora sabemos que temos no arquivo 4535 traços. Vamos ler esta informação:

```
> conta<-c(7,4,8,2,4,46,5,2,3,4,6,1,2)
> byte.me<-c(4,2,4,2,4,2,4,2,2,2,2,2,4)
> tr.bin<-c(1,2)
> l.cabe.tr.bin<-list(data.frame(campo='oi',valor=1))
> l.tr<-list(c(1,2,3))
> traco<-c(1,2,3)
> for (i in 1:num.tr){
+   tbc<-0
+   for (a in 1:13){
+     for (c in 1:conta[a]){
+       tr.bin[tbc]<-readBin(con,'int',size=byte.me[a],endian='big')
+       tbc<-tbc+1
+     }
+   }
+   frame<-data.frame(valor=tr.bin)
+   l.cabe.tr.bin[[i]]<-frame
+   for(s in 1:numero.samples){
+     traco[s]<-readBin(con,'double',size=4,endian='big')
+   }
+   l.tr[[i]]<-fl(traco)
+ }
> close(con)
```

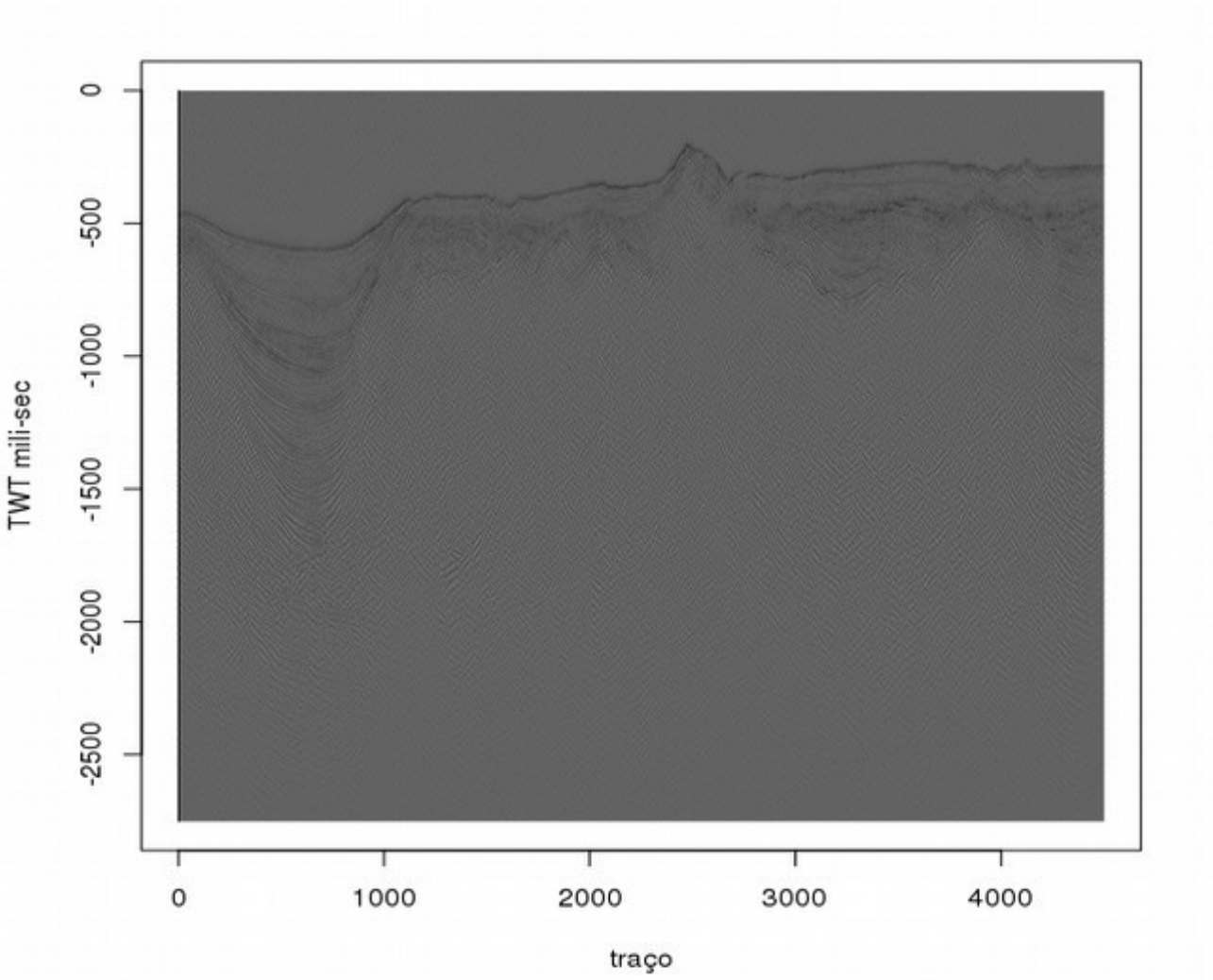
E colocamos agora toda a informação do arquivo na lista segY abaixo:

```
> segY<-list(cabe.texto,cabecalho.bin,l.cabe.tr.bin,l.tr)
```

Esta lista possui toda a informação contida no arquivo segY onde o primeiro item da lista armazena o cabeçalho texto, o segundo item armazena as informações do cabeçalho binário, o item três o cabeçalho de todos os traços individualmente e o último item cada traço individualmente.

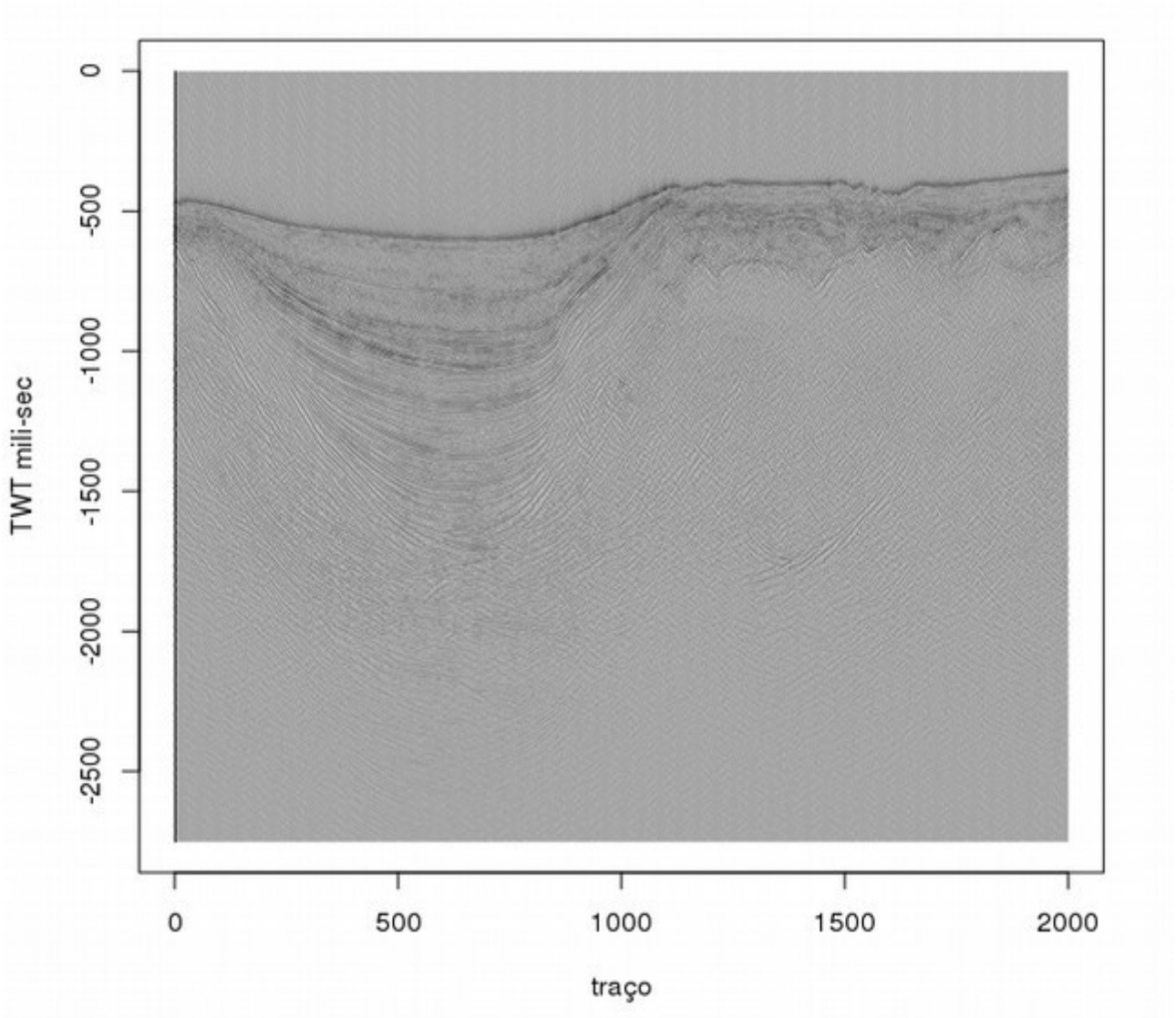
Plotando os traços da seção:

```
> plot(((segY[[4]][[1]])/10)+1,c(-1:-numero.samples),type='l',xlim=c(1,4500),
xlab='traço', ylab='TWT mili-sec')
> for(i in 2:4500){
+ lines(((segY[[4]][[i]])/10)+i,c(-1:-numero.samples),type='l',lwd=0.1)
+ }
```



Plotando os traços de 0 a 2000:

```
> plot(((segY[[4]][[1]])/10)+1,c(-1:-numero.samples),type='l',xlim=c(1,2000),
xlab='traço', ylab='TWT mili-sec')
> for(i in 2:2000){
+ lines(((segY[[4]][[i]])/10)+i,c(-1:-numero.samples),type='l',lwd=0.1)
+ }
```



Maiores informações sobre o formato estão em https://seg.org/Portals/0/SEG/News%20and%20Resources/Technical%20Standards/seg_y_rev1.pdf e nosso formato segue o descrito nesse documento.

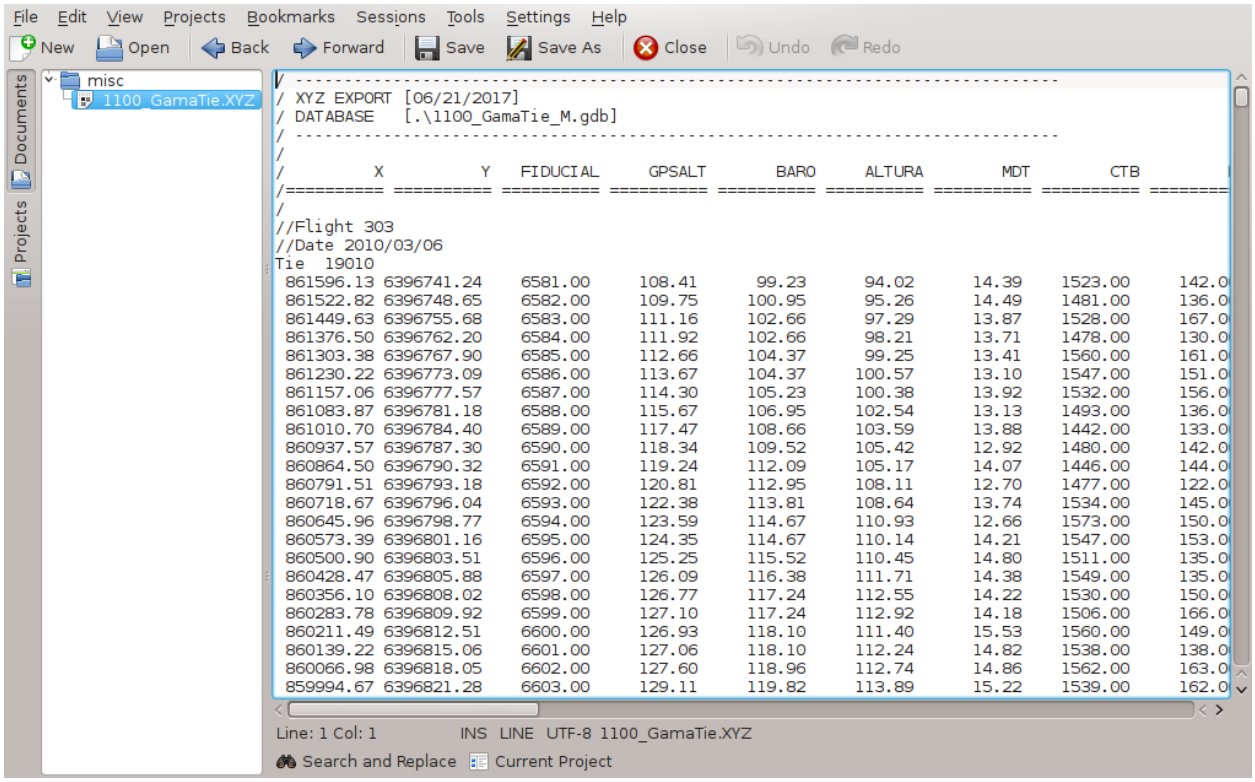
3.3.5 - Outros Dados Geofísicos_

Dados geofísicos de outra fonte tendem a seguir um mesmo formato e geralmente estão no formato texto XYZ, Para converter formatos GEOSOFT Montaj XYZ ou outros formatos XYZ para formato csv, como feito para os dados de magnetometria e gamaespectrometria acima, podemos usar um programa de planilha de dados para dados mais simples ou usar o ambiente R e criar o data.frame.

Exemplo de como usar R para ler arquivos XYZ Montaj. Primeiro vamos abrir uma conexão para o arquivo a ser lido linha por linha e descobrir o número de linhas do mesmo:

```
> con<-file('1100_GamaTie.XYZ',open='r')
> comando <- paste("wc -l 1100_GamaTie.XYZ | awk '{ print $1 }'")
> numero.linhas <- as.integer(system(command=comando, intern=TRUE))
> numero.linhas
[1] 103902
```

A imagem abaixo mostra a estrutura do arquivo que vamos ler. É importante conhecer o formato do arquivo que iremos ler.



No caso acima a sexta linha do arquivo contém o nome de cada coluna. Da décima primeira linha em diante temos os dados a serem lidos até o final do arquivo.

Vamos ler cada linha e armazenar num data.table. A linha que começa com Tie terá seu valor armazenado na última coluna.

Lendo as cinco primeira linhas:

```
> tmp <- readLines(con, n=5)
> tmp
[1] "/ ---------------------------------------------------------------------------"
[2] "/ XYZ EXPORT [06/21/2017]"
[3] "/ DATABASE   [.\\1100_GamaTie_M.gdb]"
[4] "/ ---------------------------------------------------------------------------"
[5] "/"
```

Lendo a próxima linha que contém o nome das colunas para o nosso dado. Vamos ler e criar o cabeçalho adicionando um campo ‘tie’:

```
> colunas <-c(strsplit(substring(readLines(con, n=1),11),"\\s+")[[1]],'tie')
> colunas
 [1] "X"         "Y"         "FIDUCIAL"  "GPSALT"    "BARO"      "ALTURA"
 [7] "MDT"       "CTB"       "KB"        "UB"        "THB"       "UUP"
[13] "LIVE_TIME" "COSMICO"   "TEMP"      "CTCOR"     "KCOR"      "UCOR"
[19] "THCOR"     "CTEXP"     "Kperc"     "eU"        "eTh"       "THKRAZAO"
[25] "UKRAZAO"   "UTHRAZAO"  "LONGITUDE" "LATITUDE"  "DATA"      "HORA"
[31] "tie"
> tmp <- readLines(con, n=4) # lê resto das linhas antes dos dados
```

E agora vamos ler cada linha. A que não começa com Tie será carregada normalmente no vector linha e a que contém Tie terá o valor atualizado e armazenado no final da string:

```
> linha<-c('a')
> o<-1
> tie<-''
> for (i in 11:numero.linhas){
+  l<-substring(readLines(con, n=1),2)
+  if(substring(l,1,2)=='ie'){
+    tie<-substring(l,5)
+  }else{
+    linha[o]<-paste(l,tie)
+    o<-o+1
+  }
+}
>close(con)
```

Agora armazenando os dados ordenadamente em um objeto data.table:

```
> dt<-data.table(valores=character())
> dt<-rbind(dt,list(linha))
> dtt<-dt[, c(colunas) := tstrsplit(linha, "\\s+")]
> dtt[,valores:=NULL]
> head(dtt,n=3)
          X          Y FIDUCIAL GPSALT   BARO ALTURA   MDT     CTB     KB
1: 861596.13 6396741.24  6581.00 108.41  99.23  94.02 14.39 1523.00 142.00
2: 861522.82 6396748.65  6582.00 109.75 100.95  95.26 14.49 1481.00 136.00
3: 861449.63 6396755.68  6583.00 111.16 102.66  97.29 13.87 1528.00 167.00
      UB   THB  UUP LIVE_TIME COSMICO  TEMP   CTCOR KCOR UCOR THCOR CTEXP Kperc
1: 55.00 58.00 4.00    922.00  173.00 26.70 1237.79 0.96 2.25  9.92  5.37  0.79
2: 47.00 64.00 4.00    923.00  176.00 26.70 1236.32 1.05 1.89  9.83  5.33  0.90
3: 51.00 52.00 4.00    915.00  193.00 26.70 1245.84 1.14 1.69  9.55  5.33  1.00
     eU   eTh THKRAZAO UKRAZAO UTHRAZAO    LONGITUDE     LATITUDE       DATA
1: 1.72 10.67    13.52    2.18     0.16 -53.15163955 -32.50782578 2010/03/05
2: 1.38 10.71    11.91    1.54     0.13 -53.15242107 -32.50778290 2010/03/05
3: 1.21 10.57    10.54    1.21     0.11 -53.15320127 -32.50774342 2010/03/05
          HORA   tie
1: 14:25:26.00 19010
2: 14:25:27.00 19010
3: 14:25:28.00 19010
```

Deste formato podemos trabalhar os dados tranquilamente agora.

Alguns métodos utilizam formatos proprietários mas a maioria apresenta a opção de exportar dados no formato texto (xyz) ou apresentam uma boa descrição do formato de seu arquivo de dados, quer seja texto ou binário. Entenda primeiro o formato para importar de forma correta os dados.

3.3.6 - Informações geológicas no formato de linhas e polígonos

A maior parte dos dados geológicos são representáveis na forma de pontos com atributos mas uma parte deles podem ser representados também como linhas e áreas.

Dados lineares são dados pontuais em sequência predefinida (ordenados) que podem representar um um eixo de dobra e intersecção de falhas e faturas no plano horizontal.

Um dado planar do tipo plano de acamamento, falha e fratura podem ser expressos por uma linha mas necessitam de uma informação pontual conjugada adicional representando uma direção (strike) e um mergulho (dip). Muitas vezes a informação pontual somente define este plano sem a necessidade de dado linear.

Um dado de área (polígono) é usado para definir uma unidade litológica ou domínio lito estrutural englobando um ou mais atributos.

Dados lineares e polígonos são bastante usados em dados auxiliares em geologia ligados a entidades geográficas do tipo rios, lagos, zonas urbanas, estradas, etc que dispensam análise. Para o nosso caso, são apenas camadas cosméticas.

Sempre que possível organize seus dados na forma pontal (atomizada) usando por exemplo o seguinte formato na sua planilha de dados:

ID, X , Y, Z, V1, V2, ….,Vn

onde V1 a Vn variam dependendo do tipo de informação que está sendo adquerida. Por exemplo:

Dado estrutural
ID, X, Y, Z, direção, mergulho, tipo, descrição, unidade, ordem, **domínio**

Dado de mapeamento litológico
ID, X, Y, Z, direção, mergulho, **tipo, descrição, unidade**, ordem, **domínio**

Dado de estrutura linear:
ID, X, Y, Z, direção, mergulho, tipo, descrição, **unidade, ordem, domínio**

Note que podemos usar um mesmo formato para vários propósitos mas alguns propósitos são mais únicos tais como amostragem de solo que apresenta conjunto de dados de análises químicas/mineralógicas distintas.

Sempre agrupe estas informações por domínio que poderão ajudar a definir uma área (ou volume) com o auxílio de imagens de sensoriamento remoto ou modelagem geológica.

3.4 - Considerações Finais

Tentei mostrar nesta Apostila uma visão superficial mas abrangente das possibilidades e potenciais desta nova tecnologia. Existem ramificações e especializações e cada aspecto e depende de você agora aprofundar o conhecimento e aplicação deste conhecimento. No **Volume 2** veremos a implementação de banco de dados PostGIS e a integração de dados exploratórios de geologia com outras ferramentas Open Source. ***Até lá!***