www.biologia.ufrj.br/labs/labpoly

www.r-project.org

Introdução ao uso do programa R em análises de dados ecológicos

Carlos Alberto de Moura Barboza

Paulo Cesar de Paiva

Endereço de contato: Universidade Federal do Rio de Janeiro (UFRJ), Instituto de Biologia (IB); Departamento de Zoologia; Av Prof Rodolpho Paulo Rocco s/n CCS, Bloco A, Ilha do Fundão, Rio de Janeiro, Brasil; Caixa Postal 68044, CEP: 21941-902

> este documento pode ser livremente utilizado, distribuído e reproduzido

> correções, críticas e sugestões:

> http://www.biologia.ufrj.br/labs/labpoly/

Rio de Janeiro, 2014

Use R!

1.0. R

O R é um ambiente para execução de análises estatísticas e está gratuitamente disponível na rede sob concessão GPL (*General Public License*), o que garante a todos os usuários o livre uso e distribuição, podendo ser instalado e executado em qualquer sistema operacional Windows XP (e versões posteriores), Unix, Linux e Macintosh OS X.

A linguagem R é, em grande parte, derivada da linguagem S, que na década de 80 foi amplamente difundida através do pacote comercial de estatística denominado S-PLUS. Ross Ihaka e Robert Gentleman (R, a inicial de ambos os nomes), da Universidade de Auckland na Nova Zelândia, desenvolveram uma versão reduzida da linguagem S para utilizarem em suas aulas. Ihaka e Gentleman continuaram a trabalhar no código ao longo da década de 90, o que coincidia com uma ampla difusão do sistema Linux. O R então logo se transformou em uma opção de programação para usuários Linux e a primeira versão foi lançada em fevereiro de 2000 (R version 1.0.0).

Para usuários que utilizam outras linguagens de programação o R rapidamente se tornará familiar. Entretanto, usuários dos tradicionais programas de estatística, que utilizam uma plataforma amigável de opções de "cliques", sentirão um pouco mais de dificuldade. Esta é uma etapa que na maioria das vezes inibe a adoção e utilização do R, mesmo para execução de análises básicas. Veremos que até mesmo estas são efetuadas de maneira muito mais fácil em ambiente R do que na grande maioria dos pacotes estatísticos. O entendimento da linguagem é um esforço que certamente merece ser empregado, já que uma vez aprendido, ela proporcionará uma poderosa e variada gama de opções de análises e saídas gráficas. Atualmente existem uma enorme gama de pacotes para R (http://www.r-project.org) destinados a realizar quase que qualquer análise estatística e saídas gráficas, prontas para serem incorporadas em um documento final para publicações.

1.1. Use R

Existem inúmeros motivos para se usar R em que podemos destacar: é um programa livre, logo pode ser distribuído e imune a expirações de licenças, que em casos de programas pagos podem ser caríssimas; justamente por ser livre os problemas, as ideias e inovações são amplamente discutidas na rede; como o R é uma linguagem, o programa lhe garante uma flexibilidade quase que infinita para criar e definir suas próprias funções; atualmente é a principal ferramenta para realização de análises estatísticas e está amplamente difundido em todas as universidades e centros científicos do mundo; muito provavelmente, você só

conseguirá rodar alguns modelos atualmente utilizados em análises ecológicas em R; você pode baixar, utilizar, distribuir e lecionar com o R em qualquer lugar!

Este roteiro introduz as primeiras ideias a serem entendidas para o tratamento de dados ecológicos, saídas gráficas básicas e realização de testes estatísticos. Os códigos foram compilados de diversas fontes inclusas na bibliografia deste documento, em alguns casos, desenvolvidos e/ou modificados pelos autores objetivando facilitar as rotinas regularmente utilizadas em análises ecológicas. O conteúdo deste documento pode ser livremente utilizado e distribuído, onde críticas e sugestões serão sempre muito bem vindas. Apesar de possuir algumas observações relacionadas às análises, o roteiro não objetiva explicar a álgebra empregada nas rotinas a serem realizadas. Na primeira parte será apresentado o ambiente R, como se realizar consultas, a instalação de pacotes, como citar o programa e pacotes, operações e funções matemáticas, estatística básica, a criação de vetores, matrizes e manipulação destes. Posteriormente serão abordadas as principais saídas gráficas disponíveis para análises descritivas. Em um terceiro momento abordaremos testes e análises univariadas, baseadas em modelos lineares (ex. regressões e ANOVA). Por fim, serão apresentados os códigos dos principais índices de distância, similaridade, técnicas de agrupamento, ordenamento, e análise espacial.

2.0. O ambiente R

Vamos abrir e começar a usar o programa. Como o ambiente se parece para você? Algo familiar? Observe como funciona. Por exemplo, podemos executar as funções ou qualquer comando diretamente do console. Escreva o comando e aperte Enter.

```
> 5+5

> print("olá turma")

> help(anova)

> example(lm)
```

É comum criarmos um "script", um bloco de notas de comandos onde estarão contidos todos os nossos códigos. Podemos inserir comentários ao longo de nossas análises, modificarmos e repetir tantas vezes quiser. Vá para o comando **Abrir script** (símbolo de "uma pasta se abrindo" no cantor superior esquerdo do console) e selecione **Novo script**.

Observe agora que estamos trabalhando com duas janelas, o console e o script de comandos. O script servirá como uma estória de todas as análises que desejamos. Assim podemos executar nossos comandos em uma determinada ordem. Vamos testar usar o script. Digite o mesmo comando anterior, arraste o cursor sobre o

mesmo ou, deixe-o localizado após comando, e execute **Ctrl+R** (em Mac Command+Enter).

2.1. Instalando e carregando um pacote

O R possui uma configuração básica (mesmo assim esta versão básica executa praticamente todas as funções de pacotes estatísticos convencionais do mercado). Existem diversos pacotes que podem ser diretamente instalados da rede e utilizados para se executar qualquer tipo de análise ou saída gráfica. Vamos instalar dois pacotes que nos possibilitará a, praticamente, executar todas as principais análises (univariados ou multivariados) comumente utilizadas. Clique em **Pacotes** e selecione **Instalar pacote(s)**. Escolha um espelho CRAN e depois o pacote a ser instalado. Vamos lá, instale os pacotes vegan e GAD e teste seu comando no script:

```
library(vegan) # desta forma carregamos o pacote. O símbolo de jogo da velha em programação significa comentários. O R interpretará como não sendo um comando e sim um comentário que você deseja inserir. Tente executar a função de carregar o GAD junto com um comentário

install.packages("vegan") # uma outra maneira

myPackages<-.packages(all = TRUE) # confira seus pacotes
```

A partir deste momento, você pode pesquisar sobre o pacote que executa a análise que você deseja (acredite, se não existir tudo para análises estatísticas, é quase tudo). Procure na internet, se não existir, crie!

Você precisará saber como citar o que você está usando.

```
citation("vegan")
```

Para conhecermos mais sobre o ambiente e a potencialidade do R, como exemplo a parte gráfica, vamos dar uma olhada nos principais padrões que normalmente são utilizados para a apresentação dos dados.

```
demo(graphics) # clique Enter ou com o cursor na janela dos gráficos para observar os exemplos. Observe também, que no console estão registrados os comandos que podem ser utilizados para gerar o gráfico
```

Sempre que estiver em dúvida sobre uma função peça ajuda.

```
?aov
```

Ou digite uma palavra chave e procure onde a função pode ser encontrada e como pode ser executada.

```
help.search("anova")
```

2.2. Utilizando o R como calculadora e operando funções matemáticas

O R executa todas as operações matemáticas básicas (inclusive álgebra matricial) e pode ser utilizado como uma calculadora. Vamos dar uma olhada nestas operações:

```
5+4 # somando

5*5 # multiplicando

10-2 # diminuindo

55/11 # dividindo

10^2 # exponencial

5+4;5*5;10-2;55/11 # tudo na mesma linha de comando

((4+16)/4)^2

(4^2)+(3^2)/25 ### preste sempre a atenção nos parênteses !!!

(5+5)+(10*2)-90/3
```

Qualquer função é precedida por parênteses. As funções são utilizadas com a seguinte sintaxe: `função(objeto,argumento)`.Observe:

```
log(2) # log natural

log10(2) # log na base 10

log(x,base) # escolha a base que deseja trabalhar

log(5,2)

log2(5) # atalhos para bases 2 e 10

log(2,10)#número, base

log10(2)
```

```
abs(10-20) # para obtermos o valor em módulo, “valor absoluto”. Confira a diferença retirando a função

factorial(5) # fatorial. Lembra-se da função prod? Teste ela aqui.

sqrt(4) # raiz

sqrt((3*3)^2)

prod (2,2) # produto

prod(2,2,2,2,2) # confira se é igual a 2^5

prod(1,2,3,4) # confira
```

2.3. Trabalhando com objetos

Em uma linguagem de programação devemos acessar as variáveis para o tratamento de dados. No R devemos criar objetos que representarão nossos dados, ou que serão utilizados para que operem, nos próprios dados. Estes objetos podem ser de natureza diversa como um vetor ou matriz, uma função, uma expressão, um nome de uma variável. Para criarmos um objeto utilizamos a simbologia “<-“ ou “=”. Em relação aos vetores escolhemos um nome e definimos quais são seus elementos (quantitativos ou qualitativos).

```
objeto1<-1:5

objeto1 # confira seu objeto

6:10->objeto2

objeto2

objeto3=10+10

objeto3

objeto1<-5+4; 5+5->objeto2; objeto3=10+10

objeto4<-"turma de R"
```

Como frisamos os elementos no interior dos parênteses sempre serão acompanhados por uma função que irá operar neste elemento. O R diferencia letras maiúsculas de minúsculas, logo tente utilizar uma linguagem simplificada e direta, principalmente nos nomes de objetos. Evite utilizar acentos. Podemos utilizar as funções `typeof` e `mode` para averiguarmos a natureza de um objeto.

```
mode(objeto1)
```

```
typeof(objeto4)
```

Para criarmos vetores, utilizamos a letra “c” (concatenar) e os elementos vêm entre parênteses e separados por vírgula. Quando utilizamos aspas, especificamos uma natureza categórica para os elementos. Perceba como o R interpreta a codificação para argumentos numéricos ou categóricos. Evite criar vetores compostos por elementos de natureza variada. Caso necessário, certifique-se que estará utilizando o objeto e a função de maneira correta.

```
letras<-c("a","b","c","d")

letras

numeros<-c(1:5)

numeros

numeros<-scan() # esta é uma outra maneira de criar um
vetor; digite seus elementos
```

Quando estamos trabalhando com um grande conjunto de dados, constituídos de inúmeros objetos (que podem ser vetores e/ou matrizes), podemos averiguar este universo. Ou seja, podemos listar e remover cada objeto utilizando funções para isso.

```
ls() # listando objetos

rm(objeto1) # removendo objetos

rm(list=ls())  # removendo todos os objetos listados

ls() # confira se foram removidos
```

2.4. Manipulando e editando vetores

Com os vetores já criados, podemos manipulá-los ou efetuar operações entre eles, tratando-os como objetos:

```
a<-c(1:5)

b<-c(6:10)

c<-c(a+b) # somando vetores

c

d<-c(a,b) # concatenando vetores

d
```

```
d*2

(d*2)+3
```

A função `length` determina o número de elementos de um vetor ou de "subvetores" (intervalos de valores). Os colchetes "**[ ]**" extraem uma determinada parte de um objeto. Utilizamos colchetes para designar um intervalo de elementos de um objeto. Devemos utilizar a sintaxe `objeto[intervalo]`.

```
length(d)

d[6] # extraindo um determinado valor do vetor

d[6]

d[2:5] # extraindo uma sequência de valores

d[-c(2:5)] # o que acontecerá aqui?

length(d[d>=5]) # número de elementos do objeto d que são maiores ou iguais a 5

y<-c(1.2,3.9,0.4,0.12)

y[c(1,3)] # selecione determinados valores

c(a[2:5]+b[2:5]) # criando um vetor composto por subvetores de objetos distintos

vetor<-c(6,3,5,1,8,9,5,5,5)

cummax(vetor)

cummin(vetor) # valores máximos e mínimos cumulativos de um vetor

valores<-c(1:100,20) # olhe para o último valor. Entendeu?

valores[valores <=50] # procurando valores

valores[valores>=50]

valores[valores ==0]

valores[valores!=0]

valores[valores<20|valores>25]
```

Utilizando a função `names` podemos atribuir nomes aos nossos elementos

```
x<-c(1:3)
```

```
names(x)<-c("a","b","ab")
x
x["a"]
```

2.5. Trabalhando com matrizes

Utilizamos a função `matrix` para criar matrizes e `as.matrix` para codificar um objeto como matriz. Matrizes são objetos que possuem um determinado número de linhas e colunas. O R interpreta cada linha e coluna destas matrizes como um vetor, ou seja, uma matriz de dimensão igual a 1. Existem inúmeras formas de definirmos as dimensões de nossas matrizes. As dimensões dos objetos podem ser verificadas utilizando a função `dim`. Podemos distribuir os elementos por linhas ou por colunas de dimensões predeterminadas. Se definirmos errado a dimensão de uma matriz o R retornará uma mensagem de erro (utilizando os exemplos abaixo tente averiguar na prática o que acontece). As matrizes podem ser utilizadas para qualquer operação algébrica, regressões múltiplas, autovalores, e autovetores, por exemplo.

```
x<-c(1:12)
xmat<-matrix(x,ncol=4)
xmat
matrix(x,nrow=3,byrow=T)
mat <- matrix(scan(),ncol=3,byrow=T)
```

Quando criamos matrizes os nomes das linhas e colunas são estabelecidos por números `[1]`, `[2]`, etc. Para melhor identificarmos nossos conjuntos de dados podemos nomear estas linhas e colunas.

```
rownames(xmat)<-c("lagos","rios","lagoas")
xmat
colnames(xmat)<-c("RJ","SP","SC","RS")
xmat
```

Pode ser que queiramos adicionar novas colunas à nossa matriz e para isso utilizamos a função `cbind`.

```
xmat<-cbind(xmat,c(13,14,15))
colnames(xmat)<-c("RJ","SP","SC","RS","ES")
```

```
xmat

# para adicionar linhas utilize rbind(x,y). Podemos
criar matrizes rapidamente

rbind(c(1,2,3),c(4,5,6)) # o mesmo vale para cbind
```

Suponhamos que pudessem existir 10 mil nomes de linhas e colunas que correspondem aos mesmos nomes especificados em outra matriz. Como vamos preencher isso tudo na nova matriz? Será que teremos que recorrer ao Excel para copiar e colar nomes? É claro que não!

```
y<-c(13:24)

ymat<-matrix(y,ncol=4)

ymat

colnames(ymat)<-dimnames(xmat)[[2]][1:4]

rownames(ymat)<-dimnames(xmat)[[1]][1:3]

ymat

# observe como podemos editar matrizes

y <- matrix(c(11,21,31,12,22,32),nrow=3,ncol=2)

y[2,2]<-3

y

y[2:3,] <- matrix(c(1,1,8,12),nrow=2)

y
```

Utilizamos as seguintes funções para aplicarmos operações matemáticas em linhas e colunas das matrizes:

```
rowSums(y) # soma das linhas

colSums(y) # soma das colunas

rowMeans(y) # média das linhas

colMeans(y) # média das colunas
```

2.6. Criando sequências, distribuições e realizando amostragens

Existe uma ampla variedade de formas de definir uma sequência de números, especificarmos um número de repetição desta sequência, ou parte dela em qualquer combinação que desejarmos.

```
1:10 # criando a sequência de números de 1 a 10

sum(1:10) # somando estes valores

seq(1,10)

seq(1,10,2) # criando sequencias com intervalos definidos

numeros<-c(1:3)

rep(numeros,3)

rep(numeros,1:3) # de 1 a três vezes: o primeiro uma vez, o segundo duas....

rep(1:3,each=10)
```

É bem lógico, então crie sua própria ordem de repetição.

```
rep(numeros,c(1,5,1)) # você consegue prever qual o vetor que será gerado aqui?
```

Podemos criar uma sequência de distribuição uniforme composta por n valores entre limites determinados.

```
runif(100,-10,10)
```

Da mesma maneira, podemos estabelecer uma sequência composta por um universo que possui uma distribuição normal, com média e desvio padrão predeterminado. O R possui uma grande capacidade de gerar distribuições diversas (binomial, Poisson, etc.).

```
distnorm<-rnorm(100,mean=50,sd=20)
```

Podemos sortear (função `sample`) n valores entre limites predeterminados, com reposição ou não dos elementos sorteados. Cuidado para não pedir o sorteio de um número maior de elementos do que existente em seu universo de valores! Utilize a opção `replace=TRUE` ou `FALSE` para codificar a reposição, ou não, dos elementos sorteados. Na maioria dos casos, quando não definimos a natureza de um argumento, o R utiliza o *default*. Para modificarmos a natureza lógica do argumento utilizamos "`TRUE`" ou "`FALSE`" (na maioria dos casos pode ser utilizado "`T`" e "`F`").

```
sample(x, size=y, replace = FALSE)# definindo o sorteio

sample(1:10^5, size=10, replace = FALSE)

dados<-c(1:6)

sample(dados,5,replace=TRUE)
```

Vamos jogar moedas.

```
moeda<-c("cara","coroa")

amostra<-sample(moeda,100,replace=TRUE)

(amostra[amostra=="coroa"]) # quais as jogadas do sorteio que deram coroa? Utilizamos == para especificar igualdade

(amostra[amostra!="coroa"]) # quais as jogadas do sorteio que foram diferentes de coroa? Utilizamos != para designar desigualdade

length(amostra[amostra!="coroa"]) # comece a pensar em como fazer as coisas de maneira mais elegante e funcional

# Exercite jogando dados
```

2.7 Algumas funções importantes

Para ordenar os dados utilizamos a função `sort`.

```
x<-rnorm(10)

sort(x) # utilize decresing=T para especificar a ordem de maneira decrescente

order(x) # cria postos das posições
```

Para facilitar algumas operações, como a soma das linhas e coluna de uma matriz, podemos utilizar a função `apply`.

```
som.col<-apply(xmat,2,sum) # o número dois especifica que a soma será calculada por colunas. Codifique 1 para o cálculo ser realizado por linhas
```

Crie uma lista com a função `list`.

```
a<-c(1:5)

b<-c(1:3)

lista<-list(a,b) # agora tente acessar cada elemento
da lista utilizando duplo colchetes
```

A função `sapply` organiza sua ação em um *data frame*.

```
x<-list(a=rnorm(10), b=rnorm(10))

sapply (x,mean)
```

`lapply` para obter seus resultados listados.

```
x<-data.frame("a"=rnorm(10), "b"=rnorm(10))

lapply(x,mean)
```

2.8. Criando funções

O fato de estarmos trabalhando com uma linguagem nos permite programar rotinas e criar nossas próprias funções. Esta é uma grande vantagem em relação a programas fechados, pois possuímos uma liberdade completa para tratarmos nossos dados. As funções possuem a seguinte sintaxe: `função(argumento)corpo`. A função é o nome da mesma (ou quando estamos criando a função é a palavra chave para informarmos esta atividade); o argumento são os objetos em que ela irá operar e o corpo é/são a/s ordem dada. Vamos definir uma função básica que calcula a média de um vetor.

```
media<-function(x){

     sum<-sum(x)

     n<-length(x)

     valor<-sum/n

     }

# o corpo da função vem entre chaves. Perceba que o R
não voltou nenhum resultado. Isto porque você não
pediu

media<-function(x){

     soma<-sum(x)

     n<-length(x)
```

```
    valor<-soma/n
    return(valor) # volta o valor do cálculo
    }
# é claro que existe uma função para isso...
mean(x)
media<-function(x) sum(x)/lenght(x) # como a função é
bem simples aqui ocorre o mesmo. Em funções mais
complexas cada linha deve conter um comando
```

Para programarmos rotinas lógicas podemos utilizar a função `ifelse`.

```
x<-sample(c(10:-1),20,replace=T)
ifelse(x>=0,x,print("negativo"))
```

Utilizamos a função `for` para aplicarmos uma mesma função n vezes.

```
x<-rnorm(10)
y<-rnorm(10)
xy<-vector(length=length(x))
for(i in 1:10){
    xy[i]<-cor(sample(x),sample(y))
}
xy
```

Pense em criar um código para um teste de correlação por permutação utilizando duas amostras. Mais tarde veremos como programar outras funções.

3.0. Gráficos padrões

O R produz praticamente qualquer tipo de gráfico. Entretanto, justamente em relação a este ponto, os códigos podem se tornar bastante complexos. Neste curso veremos como criar as formas gráficas básicas (gráficos de regressão, histogramas, gráficos de barras, pizza, *boxplot* e a inserção de barras de erro). Veremos posteriormente como editar gráficos relacionados aos resultados de análises multivariadas (diagrama MDS, PCA, CA, CCA, RDA). A forma mais simples de que dispomos para criar um gráfico é utilizando a função `plot`.

```
plot(sample(1:10))
```

A função `plot` também é utilizada quando queremos plotar uma variável contra a outra (útil para examinarmos a correlação entre duas variáveis).

```
plot(sample(1:10),sample(11:20))
```

Podemos mudar o tipo de símbolo e a cor dos pontos. Para isso utilizamos os argumentos `pch` (tipo de pontos, 1:20) e `col` (cor dos pontos, 1:8). Tente construir gráficos com diferentes combinações de cor e símbolo.

3.1. Gráficos em barra

Para construirmos gráficos em barra utilizamos a função `barplot`.

```
barplot(sample(1:10))
```

Vamos embelezar um pouco nosso gráfico.

```
barplot(sample(1:10), main="Título do gráfico",
xlab="Eixo x",ylab="Eixo
y",ylim=c(0,12),col=blue,las=1) # o argumento main
define o título do gráfico, xlab o título do eixo x,
ylab o título do eixo y, ylim o limite do eixo y, col
a cor das barras e las coloca a escala escrita na
vertical. Lembre que todos os argumentos que
especificam um nome na área do gráfico devem vir entre
aspas

barplot(sample(1:10),horiz=T)
```

3.2. Histogramas

Histogramas.

```
x<-rnorm(100)

hist(x)

hist(x,freq=F)

curve(dnorm(x),add=T)

hist(x,freq=F,ylim=c(0,0.5))

curve(dnorm(x),add=T)
```

3.3. Gráficos em pizza.

```
pie(c(20,30,40,50))
```

Vamos dar nomes e definir as cores dos elementos do nosso gráfico.

```
pizza<-c(1,2,3)

names(pizza)<-c("um","dois","três")

pie(pizza,col=c("black","red","blue")) # aprenda como
fazer muito mais utilizando a função example(pie)
```

3.4. Dividindo a tela

Imagine que você tenha que construir uma figura que contenha diversos gráficos. Então você poderá dividir a tela em um número de linhas e colunas que você desejar.

```
par(mfrow=c(2,1)) # a função par pode ser utilizada
para diversas funções de edição da área de plotagem.
Neste exemplo mfrow divide a tela gráfica em duas
linhas e uma coluna. Consulte sobre a função par em
seu console
```

4.0. Criando *data frames*

Um *data frame* é um conjunto de dados composto por múltiplos vetores, de mesmo número de elementos, que estão organizados em distintas colunas. Em inúmeros casos deveremos realizar nossas análises ecológicas utilizando exclusivamente *data frames*. Dados de comunidade normalmente acomodam os objetos (estações de observação) em linhas e seus descritores (espécies) em colunas. Para algumas análises as matrizes devem ser convertidas em *data frames* e tratadas como objetos.

```
aves<-c(1:5)

peixes<-c(6:10)

repteis<-c(11:15)

dados<-as.matrix(cbind(aves,peixes,repteis)) # colando
os vetores em colunas
```

```
dados<-as.data.frame(matriz)# confira a diferença na
exibição do console

dados<-data.frame("aves"=runif(10,min=1,max=30),
"peixes"=runif(10,min=1,max=70),"repteis"=runif(10,min
=1,max=20))

dados

barplot (colMeans(dataframe), col = c ( "White",
"Gray", "Black"),legend.text=T)

# exercite
```

5.0 Lendo arquivos

O R funciona como qualquer outro programa em relação a se selecionar um local de trabalho. No canto superior esquerdo do seu console, selecione **Arquivo** e posteriormente **Mudar diretório**. Desta maneira, você irá selecionar o local de trabalho onde estão contidos seus arquivos e *scripts* a serem utilizados no momento. Você pode executar a mesma função rodando um caminho como o código abaixo.

```
setwd("C:\Documents and Settings\Administrador\Meus
documentos\CursoR")
```

5.1. Lendo uma planilha de dados

Suas planilhas podem possuir a extensão .txt, .xls ou .csv (a utilização de distintas extensões de arquivos é uma característica pessoal do usuário).

Vamos utilizar a função `read.table` para ler uma planilha de dados em .txt.

```
dados<-read.table ("exe.1.txt",header=T,dec=",") # o
argumento header significa que a primeira linha da
planilha é o cabeçalho, o argumento dec informa que
decimais estão representados por vírgulas

dados<-read.table(file.choose(),h=T)

# Os dados do exercício 1 são oriundos de (Underwood,
1997, pag.423)

dados # sempre observe seus dados
```

```
names(dados) # observe o nome das variáveis de seus
dados
```

O cifrão $ é utilizado para se referir a uma determinada variável do conjunto de dados (utilize `names` para vereficar as variáveis disponíveis). No código abaixo indica que estamos nos referindo a variável tamanho do objeto dados, ou seja, do *data frame* dados. Sempre utilize $ para este tipo de objetivo. Agora confira as demais variáveis

```
dados$tamanho # inspecione cada variável. # entenda o
que significa $, são partes de seu objeto
```

A função `attach` é utilizada para se especificar o "universo de trabalho" em uso. Deste modo, podemos nos referir as variáveis sem utilizar $. Utilize esta função e inspecione as variáveis do *data frame*. Utilize `detach` para desanexar o *data frame*. Perceba o que acontece se não utilizarmos o $ para nos referirmos a uma variável, antes de utilizarmos `attach`. Tome muito cuidado quando estiver manipulando vários *data frames*! Para garantir uma menor possibilidade de erro é recomendável o uso de $ e não da função `attach`.

5.2. Exportando um *data frame*

Podemos exportar um *data frame* de volta para um arquivo de extensão .txt utilizando a função `write.table`.

```
myval <- data.frame(Fertility = c(80.2, 83.1,
92.5),Agriculture = c(17, 45.1, 39.7))

write.table(myval,"mytest.txt") # o primeiro argumento
especifica o data frame e o segundo o nome com que ele
será salvo no diretório presente

read.table("mytest.txt",header=T)
```

Podemos exportar um *data frame* para uma planilha excel. Para isso vamos utilizar o pacote `xlsReadWrite`.

```
library(xlsReadWrite)

xls.getshlib()

write.xls(myval, "mytest.xls")
```

E depois também podemos ler a planilha direto da extensão xls.

```
data<-read.xls("mytest.xls")
```

Apesar destas facilidades é recomendável trabalhar com dados em extensão de texto (.txt).

6.0. Estatística descritiva: média, variância, desvios e saídas gráficas

Vamos calcular a média de uma população utilizando algumas funções apresentadas até agora.

```
media.tamanho<-sum(dados$tamanho)/length(dados$
tamanho)

mean(dados$tamanho) # está correto?
```

a variância:

```
var_tamanho<- sum (( dados$tamanho- mean
(dados$tamanho)) ^2 / (length (dados$tamanho)-1))

var(dados$tamanho)
```

o desvio padrão:

```
dp<-sqrt(sum ( ( dados$tamanho- mean (dados$tamanho) )
^2 / (length (dados$tamanho)-1)))

sd(dados$tamanho)
```

Podemos sempre extrair informações úteis de nossos dados utilizando a função `summary`.

```
summary(dados$tamanho)

summary(dados$n_mordidas)
```

Vamos plotar um histograma das variáveis.

```
hist(dados$tamanho) # repita a operação acomodando
todos os histogramas em um único plot utilizando a
função par(mfrow=c(x,y))
```

Vamos plotar a média de cada variável utilizando a função `barplot`.

```
barplot(mean(dados)) # caso seja necessário, mude os
limites dos eixos x e y do gráfico
```

O R não possui uma função que automaticamente acrescenta a barra de erros (mas existem pacotes que fazem isso). Vamos acrescentar as barras utilizando o código abaixo.

```
?arrows # primeiramente entenda a função arrows

grafico<-barplot(mean(dados))

arrows(grafico,mean(dados)-sd(dados),
grafico,mean(dados)+sd(dados), length=0.1,angle=90,
code=3) # modifique os parâmetros length, angle e code
e observe o que acontece com o seu gráfico
```

7.0. Análises paramétricas

7.1. Testando premissas: normalidade e homocedasticidade das variâncias

Quando estamos utilizando análises paramétricas pressupomos que nossos dados possuem uma distribuição normal e homocedasticidade das variâncias (existem análises que são mais ou menos sensíveis às violações destas premissas). Utilizaremos o conjunto de dados `crabs` do pacote MASS para exercitarmos.

```
library(MASS)

crabs # data frame crabs. Consulte o manual do pacote
para entender o que significam as colunas de variáveis
e fatores
```

Vamos explorar nossos dados utilizando a estatística descritiva e exercitar o que aprendemos em sessões anteriores no que se refere à edição e tratamento dos dados. Tente utilizar a sintaxe exemplificada abaixo para se referir a cada conjunto de dados:

```
hist(crabs[[6]][1:50]) # histograma do tamanho da
carapaça dos machos da espécie B. Utilize a mesma
sintaxe para analisar qualquer variável dos grupos
distintos. Exercite criando data frames para os
diferentes grupos
```

A visualização do gráfico quantil-quantil.

```
qqnorm(crabs[[6]][1:50])

qqline(crabs[[6]][1:50])
```

Divida a tela do gráfico e plote os histogramas e o gráfico quantil-quantil em um mesmo *layout*.

Para testar a normalidade dos dados vamos utilizar o teste de Shapiro-Wilk.

```
shapiro.test(crabs[[6]][1:50]) # repita o teste para
as demais variáveis de grupos de interesse
```

Teste de Kolmogorov-Smirnov.

```
ks.test((crabs[[6]][1:50]),"pnorm",mean(crabs[[6]][1:5
0]), sd(crabs[[6]][1:50]))
```

Testando a homocedasticidade das variâncias:

```
var.test(crabs[[6]][1:50],crabs[[6]][51:100])#testando
a homogeneidade entre variâncias do tamanho da
carapaça de machos e fêmeas da espécie B
```

7.2. Teste *t* entre duas amostras

Quando as premissas de normalidade e homocedasticidade são atendidas, podemos utilizar o teste *t* para comparar a média de duas amostras independentes.

```
t.test(crabs[[6]][1:50],crabs[[6]][51:100]) # note que
o default é o teste de Welch, que pressupõe variâncias
heterogêneas. Caso queira informar que as variâncias
são homogêneas, utilize o argumento lógico
var.equal=T.

t.test(crabs[[6]][1:50],crabs[[6]][51:100],
var.equal=T)
```

Caso queiramos realizar um teste *t* para amostras dependentes, utilizamos o argumento lógico `paired=T`.

```
t.test(amostra1, amostra2, paired=T)

#teste a função utilizando o data frame anorexia do
pacote MASS
```

Suponhamos que você leu recentemente em um artigo que a média do comprimento da carapaça dos machos da espécie B era x. Podemos então testar se a média correspondente a um grupo de animais do *data frame* `crabs` difere de x.

```
t.test(crabs[[6]][1:50],mu=x)
```

Para realizar um teste unicaudal, utilizamos as opções do argumento `alternative = "less" ou "greater".`

```
t.test(crabs[[6]][1:50],crabs[[6]][51:100],
alternative = "less")
```

Antes de prosseguirmos vamos aprender a construir um belo gráfico de barras utilizando os dois fatores (espécie e sexo) dos dados `crabs`.

```
especie<-as.factor(crabs$sp)
```

```
sexo<-as.factor(crabs$sex)

means<-
with(crabs,tapply(FL,list(sexo,especie),mean,na.rm=T))

sds<-with(crabs, tapply(FL,
list(sexo,especie),sd,na.rm=T))

par(pty="s")# aqui especificamos uma área de plotagem
quadrada

grafico<-barplot(means,
beside=T,las=1,cex.axis=1.5,ann=F, ylim=c(min(0),
max(25)), xpd=F, xaxt="n",col=c("black","light
gray"),xlab="",cex.names=2.3)# pesquise sobre a função
barplot para entender os argumentos que estão sendo
utilizados

arrows(grafico, means+sds, grafico,means, ang=90,
length=0.1, code=3)

mtext(text = c("Espécie azul", "Espécie laranja"),
side = 1, line = 1.2, cex = 2,at=(grafico[2,]-0.5))

legend(locator(1),x.intersp=0.3,xjust=0,
bty="n",c("Fêmeas","Machos"), cex=1.8, fill
=c("black","light gray"))

mtext(2, text= "Tamanho do lobo frontal (mm)", line=4,
cex=2)
```

7.3. Transformações

Em alguns casos a distribuição dos dados não corresponde a uma distribuição normal e então devemos transformá-los para aplicar a estatística paramétrica.

```
log(dados) # transforma os dados para log na base
natural

log(dados+1) # quando existem valores iguais a zero

log10(dados) # atalho para bases 10 e 2

log(dados,base=5)# Escolha a base que desejar

sqrt(dados) # raiz quadrada dos dados

sqrt(sqrt(dados)) # raiz quarta dos dados
```

6.0. Testes não paramétricos entre duas amostras

6.1. Testes de Wilcoxon e Kruskal Wallis

Se ainda assim a normalidade dos dados não for alcançada, o R executa uma gama de variedades de testes não paramétricos e, no caso de duas amostras, temos a opção de utilizar o teste de Wilcoxon (*signed rank* e *rank sum*) e o de Kruskal Wallis.

```
wilcox.test(crabs[[6]][1:50],crabs[[6]][51:100]) #
equivalente ao teste de Mann-Whitney

kruskal.test(crabs[[6]][1:50],crabs[[6]][51:100])

#estes testes são equivalentes a testes t de amostras
independentes

wilcox.test(objeto, mu=x) # teste para uma média pré
determinada

wilcox.test(objeto1, objeto2, paired=T) # equivalente
a um teste t com amostras pareadas
```

7.0. Correlação e regressão

7.1. Correlação

Vamos averiguar o grau de correlação entre duas variáveis e posteriormente testar se esta relação é significativa. Para tal, vamos aplicar a análise de correlação de Pearson utilizando o arquivo exe.1.txt.

```
cor(dados$n_mordidas,dados$tamanho)

cor.test(dados$n_mordidas,dados$tamanho) # utilize o
argumento method para especificar a correlação de
"pearson" (default), "spearman" e "kendall"
```

Para a correlação entre pares de uma *data frame*

```
library(vegan)

data(varechem)

cor(varechem)

varechem
```

```
pairs(varechem)
```

7.2. Regressão linear

A análise de correlação deve ser interpretada como uma relação entre duas variáveis e não como um modelo de predição. Para este segundo objetivo, devemos utilizar a análise de regressão. Podemos definir a equação do modelo linear utilizando a função `lm`. Entenda o último código desta sessão antes de realizar as análises com seus dados reais.

```
plot(dados$n_mordidas~dados$tamanho) # vamos observar
graficamente

modelo<-lm(dados$n_mordidas~dados$tamanho) #
estabelecendo o modelo linear

abline(reg=modelo)# adicionando a linha da regressão
utilizando o modelo linear definido
```

Obtenha um resumo sobre o seu modelo linear.

```
summary(modelo) # já que o objeto do modelo já existe
```

Insira a equação do modelo linear no gráfico.

```
text(14,4.5, "y= 2.9858+0.1658*x",col="blue") # a
função text especifica um texto a ser escrito no
gráfico. O R interpreta a área do gráfico como um
plano cartesiano onde devemos especificar a posição x-
y em que o texto deve ser plotado. Neste caso, o texto
será plotado em x=14 e y=4.5. Modifique o tamanho das
letras utilizando o argumento cex e as cores
utilizando o argumento col

text(locator(1), " y= 2.9858+0.1658*x",col="black") #
a função locator especifica que você irá infomar
manualmente a posição que deve ser inserido o texto

# Entenda o teste ANOVA para regressão:

a<-c(1,2,2,2,3,4,2,4,5,8)

b<-c(1,4,2,6,3,4,2,5,9,10)

lm<-lm(a~b)

anova(lm)
```

```
summary(lm)

esperado<-predict(lm)

residuo<-resid(lm)

# y=b0+b1*x

y<-0.5995+0.5871*b

sum((esperado-mean(a))^2) # sqe da regressão

sum((a-esperado)^2) # sqe do resíduo

sum((esperado-mean(a))^2)/1 # 1 grau de liberdade

sum((a-esperado)^2)/8 # 8 graus de liberdade

pf(27.7095/1.2988,1,8,lower.tail=F)

b0<-0.5995

b1<-0.5871

# exercite a correlação com o data frame Animals do
pacote MASS
```

8.0. Análise de variância (ANOVA)

A ANOVA é uma das principais ferramentas utilizadas em análises de dados ecológicos e está disponível em todos os pacotes estatísticos. A função básica do R para realizar a ANOVA é `aov`, especificando o modelo linear, um objeto do tipo `lm`. Vamos utilizar o data frame `crabs` para exercitar a análise ANOVA.

8.1. Testando premissas da ANOVA

Teste de normalidade para a variável `crabs$FL` do *data frame* `crabs` do pacote `MASS`.

```
shapiro.test(crabs$FL)  # os dados possuem uma
distribuição normal?
```

Vamos utilizar um modelo linear unifatorial (fator espécie com dois níveis, espécies O e B). Teste a homogeneidade das variâncias utilizando o teste de Barlett.

```
bartlett.test(crabs$FL~especie, data=crabs)
```

Testando a homocedasticidade utilizando o teste de Cochran. Para realizar o teste e prosseguirmos com as análises posteriores vamos utilizar o pacote GAD.

```
library(GAD)

especie<-as.fixed(crabs$sp)# definindo a espécie como
um fator fixo

modelo<-lm(crabs$FL~especie, data=crabs)

C.test(modelo)
```

8.2. ANOVA Fatorial

Primeiramente, vamos dar uma olhada no gráfico *boxplot* da variável lobo frontal das duas espécies, e somente depois utilizar a ANOVA para testar a diferença entre as médias.

```
boxplot(crabs$FL~especie, data=crabs)
```

Exemplo de como se utilizar a função `tapply`.

```
barplot(tapply(crabs$FL, crabs$sp,mean))
```

Definindo o modelo linear e rodando a ANOVA.

```
modelo<-lm(crabs$FL~especie) # definindo o modelo
linear

tab.anova<-anova(modelo) # objeto da análise ANOVA do
modelo

tab.anova # ou somente anova(modelo) para visualizar o
resultado
```

Podemos utilizar a função `aov`.

```
aov(crabs$FL~especie,data=crabs)

summary(aov(crabs$FL~especie,data=crabs))
```

Agora vamos considerar o sexo como um fator. O R possui uma função lógica para verificarmos se isto está sendo considerado.

```
is.factor(crabs$sex)
```

Qual a natureza do objeto?

```
mode(crabs$sex)
```

Codifique que o sexo é um fator.

```
sexo<-as.factor(crabs$sex)
```

Vamos verificar graficamente se os fatores possuem interação,

```
interaction.plot(crabs$sp,crabs$sex,crabs$FL)
```

ou podemos utilizar

```
interaction.plot(especie,sexo,crabs$FL)

interaction.plot(crabs$sex,crabs$sp,crabs$FL)
```

Vamos entender o teste ANOVA comparando os resultados dos modelos uni e bifatorial.

```
modelo.sex<-lm(crabs$FL~sexo,data=crabs) # modelo
unifatorial, fator=sexo

modelo.sp<-lm(crabs$FL~especie,data=crabs)# modelo
unifatorial, fator=especie

modelo.sp.sex1<-lm(crabs$FL~sexo+especie,data=crabs) #
modelo bifatorial, fatores=sexo,especie; não ocorrendo
interação

modelo.sp.sex2<-lm(crabs$FL~sexo*especie,data=crabs) #
modelo bifatorial, fatores=sexo,especie; com interação

anova(modelo.sex)

anova(modelo.sp)

anova(modelo.sp.sex1)

anova(modelo.sp.sex2)
```

Vamos utilizar o teste *a posteriori* de Tukey para investigar quais os pares de médias que diferem entre si.

```
modelo.aov<-
aov(crabs$FL~sexo*especie,data=crabs)tukey<-
TukeyHSD(modelo.aov) # a função TukeyHSD deve ser
executada utilizando-se um objeto do tipo aov

tukey
```

Perceba que até o momento não definimos a natureza dos fatores que estamos testando. Preste atenção nas razões *F* que estão sendo calculadas.

8.3. Modelos mistos e aninhados

Em muitos pacotes estatísticos ficamos impossibilitados de adotar modelos lineares mais complexos. Lembre-se que o modelo linear estabelecido em análises

com dois ou mais fatores (podendo ser fixos ou aleatórios), define as razões *F* a serem calculadas. Vamos verificar isto na prática. Utilizando o pacote GAD vamos codificar os fatores sexo e espécie como fixos, como aleatórios e um como fixo e outro como aleatório (um modelo misto).

Modelo de fatores aleatórios.

```
especie.r<-as.random(crabs$sp)

sexo.r<-as.random(crabs$sex)

modelo.random<-
lm(crabs$FL~sexo.r*especie.r,data=crabs)
```

Modelo de fatores fixos.

```
especie.f<-as.fixed(crabs$sp)

sexo.f<-as.fixed(crabs$sex)

modelo.fixed<-lm(crabs$FL~sexo.f*especie.f,data=crabs)
```

Modelo com um fator fixo e outro aleatório.

```
sexo.r<-as.random(crabs$sex)

modelo.mixed<-lm(crabs$FL~sexo.r*especie.f,data=crabs)
```

Vamos comparar os modelos.

```
gad(modelo.random)

gad(modelo.fixed)

gad(modelo.mixed)
```

Utilize a função `estimates` para conferir as razões F que estão sendo calculadas.

```
estimates(modelo.random)
```

Vamos definir um modelo aninhado: espécie é um fator aleatório com dois níveis (B e O) aninhado no fator fixo sexo com dois níveis (M e F). Segundo Underwood (1997) somente fatores aleatórios podem ser aninhados, contudo esta afirmação não é unanimidade. O modelo definido é dependente da hipótese a ser testada, portanto a interpretação biológica dos resultados depende de uma proposição correta da hipótese e do modelo linear.

```
modelo.nested<-
lm(crabs$FL~sexo.f+especie.r%in%sexo.f,data=crabs)
```

```
gad(modelo.nested) # preste atenção nos fatores da
tabela e nas razões F
```

9.0. Estatística multivariada

O conjunto de dados ecológicos a ser utilizado em análises multivariadas, tradicionalmente está organizado em matrizes. Este *data frame* possui *n* x *p* dimensões e está disposto de forma a acomodar os objetos nas linhas e seus descritores (variáveis) em colunas. Deste modo, análises de agrupamento e ordenação geram matrizes associativas quadradas e simétricas, ou seja, contendo todas as combinações par a par, de similaridade ou distância, estipuladas por um determinado índice. O índice que mede a distância ou similaridade entre os objetos x-y é igual ao valor entre y-x.

9.1. Diversidade e riqueza

O R calcula as principais medidas de diversidade utilizadas para matrizes de comunidade. Vamos carregar o vegan e acessar o *data frame* BCI.

```
library(vegan)

data(BCI)

BCI

?BCI # entenda o data frame

div.shan<-diversity(BCI, index = "shannon", MARGIN =
1, base = exp(1)) # a diversidade de Shannon. MARGIN=1
codifica que ela deve ser calculada por linhas

div.simp<-diversity(BCI, index = "simpson", MARGIN =
1, base = exp(1))

par(mfrow=c(2,1))

barplot(div.shan)

barplot(div.simp)

?diversity # pesquise a função e faça os gráficos para
outros índices
```

Vamos consultar Landeiro (2011) e exercitar a função de diversidade de Shannon criada pelo autor.

```
shannon<-function(dados){
     prop<-dados/sum(dados)
     resu<-numeric()
     n<-length(prop)
     for(i in 1:n){
     resu[i]<-if(prop[i]>0){prop[i]*log(prop[i])}
     else{0}
}
     H<- -sum(resu)
     return(H)
}

shannon(as.matrix(BCI[1,])) # confira com o resultado
da sessão anterior
```

Calculando a rarefação de Hulbert das amostras

```
raref<-rarefy(BCI[1,],1:sum(BCI[1,]),se=T)     #     a
rarefação da primeira localidade. se=T codifica que
deve ser calculado o erro padrão. Calcule para as
demais localidades

rarecurve(BCI) # plote as curvas das localidades do
data frame. É bom para darmos uma primeira olhada nos
dados, verificar se as curvas de mesmas localidades
possuem o mesmo padrão

rarecurve(BCI[1:10,]) # escolha quais devem ser
plotadas

# Exercite plotando curvas com diferentes símbolos e
cores e inclua os desvios de cada uma

plot(raref[1,],col="red",type="n")

lines(raref[1,],col="red")
```

Podemos utilizar estimadores não paramétricos para extrapolar o número de espécies presentes na comunidade. Vamos utilizar a função `poolaccum` para plotar curvas assintóticas de riqueza específica (ex. Chao, Jacknife 1 e 2 e Bootstrap).

```
estimate<-poolaccum(BCI, permutations = 999)
```

```
plot(estimate$chao[,2],xlab="n amostras",ylab="n
espécies",type="n")

lines(estimate$chao[,2]) # usamos $ pois o objeto
estimate é uma lista. Confira utilizando summary

lines(estimate$jack1[,2],col="red")
```

9.2. Coeficientes de dissimilaridade e similaridade

Existem incontáveis índices associativos métricos, não métricos, simétricos ou assimétricos, que podem ser utilizados para se definir uma matriz de associação (distância ou similaridade) em modos *Q* (entre objetos) e *R* (entre espécies). Estes índices, muitas vezes, possuem naturezas distintas e devem ser utilizados em situações particulares. Por exemplo, existem determinados índices ou coeficientes que são mais apropriados para uma matriz de dados biológicos (ex. Bray-Curtis) e outros mais apropriados para variáveis ambientais (ex. distância Euclidiana). É importante termos em mente as seguintes perguntas: 1) Estamos utilizando o modo *Q* ou *R*? 2) Estamos tratando de abundância de espécies ou algum outro tipo de variável? 3) Os dados são binários, quantitativos ou de ambas as naturezas?

No R todas as medidas de similaridade são automaticamente transformadas para distância. Dependendo do pacote, a distância é computada como D=1-S ou D=raiz(1-S). É importante estarmos certos de qual transformação foi utilizada devido à natureza métrica ou não métrica de D calculado.

A distância Euclidiana é default.

```
data(varechem)# consulte o manual do vegan para as
informações deste data frame

eucl<-dist(varechem) # a distância Euclidiana é
default

eucl
```

A distância de Bray-Curtis é uma das distâncias mais utilizada em matrizes de dados biológicos quantitativos.

```
dados.bray<-vegdist(BCI) # Bray-Curtis é a distância
default do vegan

dados.bray

bray<-vegdist(log(BCI)) # caso queiramos computar a
distância dos dados logaritimizados
```

Preste atenção na função para o cálculo de similaridade (não dissimilaridade como no vegan) Bray-Curtis extraída de Landeiro (2011).

```
bray<-function(dados.spp){
     n<-nrow(dados.spp)
     BrayCurtis<-matrix(NA,n,n)
     for(i in 1:n){
     for(j in 1:n){
     numerador<-sum(abs(dados.spp[i,]-dados.spp[j,]))
     denominador<-sum(dados.spp[i,]+dados.spp[j,])
     BrayCurtis[i,j]<- numerador/denominador
}
}
     return(as.dist(BrayCurtis))
}
```

A distância de Chord é a distância euclidiana dos dados normalizados.

```
dados.norm<-decostand(BCI, "nor")
dados.chord<-dist(dados.norm)
```

Vamos criar uma função para normalizar os dados.

```
norm<-function(dados){
     nl<-nrow(dados)
     nc<-ncol(dados)
     vetor<-numeric()
for(i in 1:nl){
     vetor[i]<-sqrt(sum(dados[i,]^2))
}
matriz<-matrix(NA,nl,nc)
for(i in 1:nl){
     matriz<-dados/vetor
```

```
}

return(matriz)

}

norm(dados) # confira com a função do vegan se foi calculado corretamente
```

A distância de Hellinger é a distância euclidiana dos dados transformados por Hellinger.

```
dados.hel<-decostand(BCI, "hel")

dist.hel<-dist(dados.hel)
```

Quando desejamos trabalhar com dados binários podemos criar esta matriz binária.

```
dados.pa<-decostand(BCI, "pa")
```

Para utilizar coeficientes de distância (1-S) apropriados para dados binários, codificamos `binary=T`.

```
dados.sor<-vegdist(BCI,binary=T) # 1-S de Sorensen é a dissimilaridade default para dados binários

dados.jac<-vegdist(BCI,"jac",binary=T) # dissimilaridade de Jaccard
```

ou

```
dist(dados,"binary")
```

A distância de Ochiai é o valor da distância de Chord ou Hellinger calculados para dados de presença, divididos pela raiz de 2. O cálculo produz (raiz(1-S)).

```
dados.ochiai<-dist.binary(BCI,method=7)

?vegdist # pesquise todas as possibilidades existentes para se calcular as matrizes de distância
```

9.3. Métodos de agrupamento (hierárquicos e não hierárquicos)

As análises de agrupamento objetivam apresentar uma disposição estruturada e simplificada da relação entre objetos e descritores, visando reconhecer descontinuidades normalmente representadas por sinais ou, por gradientes contínuos, como ocorre na maioria dos dados ecológicos. Lembre-se que, em princípio, não estamos testando hipóteses preestabelecidas, e sim empregando métodos algébricos investigativos.

Single Linkage Aglomerative Cluster - Revela gradientes de maneira eficiente, contudo dificulta o reconhecimento de partições.

```
BCI.norm<-decostand(BCI,"normalize") # para análises de agrupamento também podemos utilizar os dados transformados por range

BCI.dist<-vegdist(dados.norm,"euc")# a distância de Chord

cluster.single<-hclust(BCI.dist,method="single")

plot(cluster.single)
```

Complete Linkage Aglomerative Cluster - Ideal para se identificar descontinuidades.

```
cluster.complete<-hclust(BCI.dist,method="complete")

plot(cluster.complete)
```

Average Aglomerative Cluster - Normalmente os dendrogramas gerados são uma série de grupos aninhados.

```
cluster.average<-hclust(BCI.dist,method="average") # para a distância média

plot(cluster.average)

cluster.centroide<-hclust(BCI.dist,method="centroid") # para a distância da centróide do grupo

plot(cluster.centroide)
```

Wards's Minimum Variance Clustering - O algoritmo minimiza a soma dos erros quadráticos intra-grupos, que é a soma das distâncias entre os pares de membros divido pelo número de elementos do grupo.

```
cluster.ward<-hclust(BCI.dist,method="ward")

plot(cluster.ward)
```

Por se tratarem de métodos heurísticos, a escolha do algoritmo de agrupamento influencia o resultado final. A distância cofenética entre dois objetos no dendograma é a distância que os une a um grupo comum. Para decisão de qual método utilizar, podemos regredir as matrizes de distância criadas contra as matrizes cofenéticas do método de agrupamento utilizado.

```
BCI.cof.sing<-cophenetic(cluster.single)
```

```
cor(dados.dist,BCI.cof.sing)
```

Calcule a correlação para todos os métodos utilizados, verifique seus valores e plote um dendrograma final.

```
dendro<-as.dendrogram(cluster.ward)

plot(dendro,horiz=T)
```

Non-hierarchical cluster - K-Means - A análise *K-means* procura identificar o número *k* grupos de partições definidas, formadas por objetos que são mais semelhantes entre si do que em relação a qualquer outro grupo distinto. O procedimento minimiza a soma dos erros quadráticos intra-grupos, e é o mesmo utilizado no método aglomerativo de Ward. *K-means* é um método linear, logo a distância Euclidiana está implicitamente embutida na análise. Importante: Para transformações onde não podemos aplicar diretamente a distância euclidiana (ex. Bray-Curtis) a matriz analisada deve ser a obtida através de coordenadas principais.

```
BCI.norm<-decostand(BCI,"norm") # poderíamos também
utilizar a transformação de Hellinger

spe.kmeans<-kmeans(BCI.norm,centers=3,nstart=100) #
escolhendo três grupos, rodando 100 vezes
```

Podemos plotar os grupos nas posições geográficas em que se encontram. Vamos utilizar as coordenadas do data frame BCI para fazer isso.

```
UTM.EW <- rep(seq(625754, 626654, by=100), each=5)

UTM.NS <- rep(seq(1011569, 1011969, by=100), len=50)

plot(UTM.EW,UTM.NS,type="n")

text(UTM.EW,UTM.NS,labels=spe.kmeans$cluster)

BCI.kmeans.grup<-cascadeKM    (BCI.norm,    inf.gr=2,
sup.gr=8,iter=100,criterion="ssi") # rodando a análise
para escolher entre 2 a 8 grupos

plot(spe.kmeans.grup,sort=T) # representação gráfica
da análise

barplot(spe.kmeans.cascata$results[2,])

plot(UTM.EW,UTM.NS,type="n")

text(UTM.EW,UTM.NS,labels=BCI.kmeans.grup$partition[,5
]) # plote a figura novamente com quantos grupos você
```

```
quiser. Exercite plotando símbolos diferentes para
cada grupo.
```

9.4. Métodos de ordenação simples (*unconstrained*)

A grande maioria dos métodos de ordenação é baseada em análises de autovalores e autovetores (*eigenanalysis*). Uma das exceções é a análise não métrica multidimensional (nMDS). Métodos de ordenação buscam descrever a distribuição dos objetos no espaço multidimensional. Normalmente, este número de dimensões é aproximadamente igual ao número de variáveis presentes na matriz. Como a disposição dos grupos é formada por nuvens de pontos, estas possuem uma maior variabilidade em direção a um determinado eixo, e esta variabilidade é estimada pelo respectivo autovalor. A análise irá extrair a variabilidade concentrada em outra direção, ortogonal a primeira, e a representará ao longo da componente principal seguinte. O primeiro autovalor expressa a maior variabilidade, o segundo autovalor expressa a variabilidade contida ao longo da segunda componente e assim por diante. A análise prossegue até que toda a variabilidade contida nos dados seja expressa sob a forma de componentes ortogonais. Portanto o valor de cada autovetor expressa a posição de cada objeto no referida componente.

Principal Components Analysis (PCA). Largamente utilizada para ordenação de dados ambientais. A PCA preserva a distância euclidiana e por isso é extremamente sensível a matrizes contendo um grande número de zeros. Este é o principal motivo pelo qual a análise é evitada para ordenar dados biológicos. Entretanto, utilizando a transformação adequada (transformações que preservem a distância Euclidiana entre os pontos e que não é influenciada pela presença de duplos zeros) a PCA é uma análise eficiente para dados biológicos. É recomendável utilizar a matriz de covariância quando estamos tratando de descritores de mesma ordem de magnitude e possuem mesmas unidades. Por outro lado, devemos utilizar a matriz de correlação quando estamos tratando de descritores de natureza distinta.

```
pca<- prcomp(BCI,scale=F)  # scale=T codifica que a
análise será realizada utilizando-se a matriz de
correlação, ou seja, as variáveis serão padronizadas.
Com scale=F a análise será efetuada com a matriz de
covariância.

summary(pca)

biplot(pca,scaling=1)# scaling codifica o tipo de
projeção do biplot. Consulte Legendre & Legendre
(1998) para entender o significado deste argumento

biplot(pca,col=c("gray","black")) # mude as cores do
plot
```

```
# exercite modificando os argumentos scaling e scale
```

Podemos rodar a análise utilizando a função `rda` do pacote vegan.

Vamos exercitar criando a função para análise de componentes principais. O código foi retirado de Bocard et al., (2011); pag. 150.

```
pca<-function(Y){

    dados<-as.matrix(Y)

    objetos<-rownames(Y)

    variaveis<-colnames(Y)

    dados.cent<-scale(dados,center=T,scale=F) # aqui
    estamos centrando os dados

    dados.cov<-cov(dados.cent) # aqui extraímos a
    matriz de covariância dos dados centrados

    eigen.dados<-eigen(dados.cov)

    U<-eigen.dados$vectors # aqui criamos a matriz U
    contendo os autovetores da matriz de covariância.
    Estas são as coordenadas das variáveis no biplot

    rownames(U)<-variaveis

    F<-dados.cent%*%U # eq. 9.4 em Legendre e
    Legendre (1998). Estas são as coordenadas dos
    objetos no biplot

    U2<-U%*%diag(eigen.dados$values^0.5) # aqui
    calculamos as coordenadas das variáveis para
    scaling=2

    rownames(U2)<-variaveis

    G<-F%*%diag(eigen.dados$values^0.5) # e então as
    coordenadas dos objetos, scaling=2

    rownames(G)<-objetos

    resultado<-list(eigen.dados$values,U,F,U2,G)

    names(resultado)<-c("autovalores","U",

    "F","U2","G")

    resultado
```

```
}

pca(dados)# teste a função

plot.pca<-pca(dados)

biplot(plot.pca$F,plot.pca$U,xlab="CP1",ylab="CP2")
```

Principal Coordinate analysis (PCoA). Como anteriormente já citado, não é recomendável utilizar matrizes com grandes quantidades de zeros para a ordenação de componentes principais. Já a ordenação por coordenadas principais parte de uma matriz de distância gerada por qualquer medida. Computar uma PCoA para uma matriz de distância Euclidiana equivale a realizarmos uma análise PCA utilizando a matriz de covariância e plotando o gráfico com `scaling=1`.

```
data(dados)

dados.bray<-vegdist(dados)

spe.pcoa<-cmdscale(dados.bray,k=(nrow(dados)-1),eig=T)
# observe que os últimos autovalores são negativos,
logo não interpretáveis do ponto de vista do plano
Euclidiano

ordiplot(spe.pcoa$points[,c(1,2)],type="t",main="PCoA"
, xlab="eixo 1",ylab="eixo 2")

abline(h=0,lty=3) # como os dados foram centrados para
realização da análise, podemos traçar estas linhas
originárias da origem.

abline(v=0,lty=3)

scores(spe.pcoa) # conferindo os scores dos objetos.
Veja que são os mesmos valores de spe.pcoa$points

spe.names<-wascores(spe.pcoa$points[,1:2],dados)

text(spe.names,rownames(spe.names),cex=0.7,col="red")
# atribuindo os nomes e plotando as espécies
```

Non-metric Multidimensional Scaling (n-MDS). A n-MDS não é uma análise métrica e não utiliza autovalores e autovetores para a ordenação dos objetos (portanto, a variabilidade não é maximizada ao longo dos eixos). O objetivo é acomodar os objetos em um número pequeno de eixos predeterminados, de modo a melhor representar a relação de distância entre eles. O procedimento parte de uma organização inicial e, interativamente, reorganiza os objetos no plano dos eixos de maneira a diminuir o *stress*. O *stress* é uma função que informa a magnitude da perda de informação das novas distâncias em relação às distâncias

originais da matriz de associação. O método é largamente utilizado por ecólogos, já que admite a utilização de inúmeros coeficientes não métricos, como o de Bray-Curtis.

```
dados<-crabs

# edite o data frame para ficar somente com as variáveis e o número dos indivíduos

dados<-cbind(crabs[,4:8])

mds<-metaMDS(dados, distance = "bray", k = 2, trymax = 500)

summary(mds)

plot(mds)

plot(mds, type=”text”, cex=1.2)

plot(mds$points)

# vamos plotar grupos com diferentes símbolos

pontos<-as.data.frame(mds$points)

xrange<-range(pontos[,1])

yrange<-range(pontos[,2])

plot(pontos[1:100,],xlim=xrange,ylim=yrange,cex=2,col="blue",pch=19)

points(pontos[101:200,],xlim=xrange,ylim=yrange,cex=2,col="orange",pch=19)

legend(locator(1),c("blue crab","orange crab"), cex=1,pch=c(19,19),col=c("blue","orange"))

text(locator(1),"stress=0.08",cex=1)

# exercite plotando mais variáveis,por exemplo, além da espécie por sexo também
```

Correspondence Analysis (CA). Como a análise n-MDS, a CA é uma ferramenta frequentemente utilizada. Isto porque, o procedimento não é influenciado pela grande quantidade de zeros. A álgebra utilizada consiste em primeiramente transformar a matriz biológica em uma nova matriz $\bar{Q}$, preservando a distância qui-quadrado, para posterior extração de autovalores e autovetores. Esta é uma análise recomendada para representar gradientes ambientais completos.

```
dados<-as.data.frame (read.table (file.choose() , row.names=1 , dec=","))

spe.ca<-cca(dados)

summary(spe.ca)

dados.log<-log(dados+1)

spe.ca<-cca(dados.log)

plot(spe.ca,type="text")

plot(spe.ca, type="n")

text(spe.ca,col="red",cex=1.2,labels=rownames(dados))

text(spe.ca, "species", col="blue", cex=0.8)
```

Detrended Correspondence Analysis (DCA). Em gradientes ambientais muito extensos os locais correspondentes aos extremos, normalmente, não possuem espécies em comum. Esta é a justificativa para se preservar a distância qui-quadrado entre objetos ao invés da distância linear. Quando isto ocorre, a ordenação resultante normalmente assume a forma de um arco, e o ruído é classicamente denominado por "efeito de arco". A DCA foi desenvolvida para se evitar este ruído, contudo é um procedimento que possui inúmeros problemas e deve ser evitado. A rotina está disponível no pacote vegan através da função `decorana`.

```
spe.dca<-decorana(dados.log)

summary(spe.dca)

plot(spe.dca)
```

9.5. Métodos de ordenação canônicos (*constrained*)

Ordenações canônicas resultam da relação entre duas matrizes, uma matriz resposta (geralmente representada por um conjunto de dados bióticos) e uma matriz explicativa (pode ser representada por dados bióticos e abióticos); são combinações de métodos de ordenação e regressão múltipla. Nesta seção abordaremos os códigos das duas principais ordenações canônicas utilizadas em análises ecológicas: Análise de Redundância (RDA) e Análise de Correspondência Canônica (CCA). Veremos que rodar estas análises no R é muito mais simples e

rápido do que utilizar o programa Canoco. Os códigos utilizados nesta seção seguem o protocolo apresentado por Borcard *et. al*., (2011).

***Redundancy Analysis* (*RDA*)**. O termo redundância é sinônimo de "variabilidade explicada". A análise deve ser empregada quando queremos explicar a ordenação de uma matriz Y em função de variáveis explicativas X. É uma extensão da regressão múltipla, e cada eixo canônico é uma combinação linear da matriz explicativa, preservando-se a distância Euclidiana entre os objetos no diagrama de ordenação. Justamente por este motivo, a RDA não foi primariamente tão disseminada na Ecologia, quanto foi a CCA. Contudo, com a adoção de transformações adequadas para matrizes biológicas (ex. Distância de Hellinger) a RDA é um dos mais robustos e eficientes métodos de ordenação canônica.

```
varespec.hel<-decostand(varespec,"hellinger")

varechem.sdz<-decostand(varechem,"standardize")#
análises canônicas normalmente devem ser precedidas
pela padronização da matriz explicativa

rda<-rda(varespec.hel~.,varechem.sdz)

summary(rda)
```

A proporção da variabilidade explicada pela combinação linear das variáveis explicativas é equivalente ao R^2 da regressão múltipla, portanto, deve também ser corrigido.

```
(R2.adj<-RsquareAdj(rda)$adj.r.squared)
```

Como no Canoco, podemos testar através de permutações a significância do modelo Global ou testar a significância da variabilidade explicada por cada eixo canônico.

```
anova.cca(rda, step=1000) # step codifica o número de
permutações

anova.cca(rda, step=1000, by= "axis") aqui testamos a
significância de cada eixo
```

É interessante acessarmos o coeficiente canônico das variáveis.

```
coef(rda) # equivale ao coeficiente de regressão de
cada variável em relação a cada eixo
```

O *triplot* e o *biplot* dos resultados.

```
plot(rda,type="text") # o argumento type codifica o
plot para os nomes contidos no data frame
```

```
plot(rda,type="text",display=c("cn","wa")) # utilize display para especificar o que deve ser plotado: wa(weighted sum) para os objetos no plano das variáveis resposta; lc(linear combination) para os objetos no plano das variáveis explicatórias; cn(constrained) os vetores das variáveis explicatórias; sp(species) para as espécies
```

Para um controle da edição do que deve ser plotado.

```
plot(rda, type="n")

text(rda, dis="cn")

text(rda, cex=1.2)

text(rda, "species", col="blue", cex=0.8)

# uma outra maneira mais liberal

par(pty="s")

plot(rda, type="n")

text(rda, dis="cn")

sp.sc<-scores(rda,choices=1:2,scaling=1,display="sp")
# aqui criamos uma data frame contendo os scores das espécies

wa.sc<-scores(rda,choices=1:2,scaling=1,display="wa")
# os scores dos objetos (weighted average)

# agora utilize o código de exemplo da seção de nMDS para plotar os pontos da maneira que desejar

points(wa.sc[1:13,],pch=21,col="black",bg="green",cex=2) # se quiser codificar as espécies com símbolos e cores diferentes. Se quiser plotar os nomes use a função text, como exemplificado na seção anterior. Plote os demais pontos com símbolos e cores diferentes

points(wa.sc[14:24,],pch=22,col="black",bg="blue",cex=2)

legend(locator(1),c("grupo1","grupo2"),cex=2,
pch=c(21,22),pt.bg=c("green","blue"),bty="n") # inserindo a legenda. Plote uma ordenação para as espécies e outra para os objetos em um mesmo plot
```

Vamos tornar a explicação do nosso modelo mais simples, no que resultará em uma exposição gráfica mais limpa. Através de permutações, testemos a significância de cada variável de maneira independente.

```
step.forward<-
ordistep(rda(varespec.hel~1,data=varechem.sdz),scope=f
ormula(rda),direction="forward", pstep= 1000) # agora
teste com direction="backward"


ordistep(rda(dados.hel~1,data=as.data.frame(pcnm$vecto
rs)),scope=formula(rda),direction="forward",pstep=1000
)
```

A seleção *a posteriori* baseada nos valores de significância de α pode, em alguns casos, selecionar variáveis que não necessariamente possuem uma relação significativa com o modelo, inflando a probabilidade de erro do Tipo I e superestimando R^2. Vamos utilizar o programa `packfor` e aplicarmos o código para contornar estes problemas.

```
library(packfor)

(R2a.all<-RsquareAdj(rda)$adj.r.squared)

forward.sel(varespec.hel,varechem.sdz,adjR2thresh=R2a.
all) # Compare com os resultados obtidos pela função
ordistep
```

Crie uma *data frame* com as variáveis selecionadas (“modelo parcimonioso”) e plote o gráfico.

***Canonical Correspondence Analyses* (*CCA*)**. A CCA é uma extensão da CA e os procedimentos são bastante semelhantes à RDA. Entretanto, a CCA não preserva uma distância linear e sim uma distância qui-quadrado entre os objetos. Desde a década de 80, a análise se tornou amplamente utilizada na Ecologia. Contudo, o procedimento parte de duas premissas que são difíceis de serem testadas ou até mesmo investigadas: 1) as espécies foram amostradas em todo o seu gradiente de distribuição; 2) a distribuição das espécies responde a um modelo unimodal em relação aos principais parâmetros ecológicos que regulam esta distribuição. A CCA é extremamente afetada por espécies raras, já que a contribuição da diferença da abundância destas é superior à contribuição da diferença entre as espécies comuns. É aconselhável excluir as espécies raras antes de rodar a análise (isto vale também para CA). Apesar de amplamente utilizada, a distância qui-quadrado possui inúmeras restrições algébricas e muitas vezes não condiz com a realidade dos dados ecológicos. A inércia explicada não pode ser corrigida adequadamente e, portanto, a probabilidade de erro do Tipo I e a magnitude da variabilidade

explicada podem estar superestimadas. O código empregado para CCA é extremamente parecido com o que utilizamos para RDA.

```
cca<-cca(varespec~.,varechem.sdz) # não utilize a transformação de Hellinger. Podemos utilizar log

summary(cca)

plot(cca,scaling=2,display=c("sp","lc","cn"),type="text",cex=0.8)

par(mfrow=c(1,2)) # plote as espécies e os objetos em gráficos separados

plot(cca,scaling=1,display=c("cn","sp"),type="text",cex=0.8)

plot(cca,scaling=1,display=c("cn","lc"),type="text",cex=0.8) # ou plote wa
```

Assim como na RDA, podemos testar a significância global da análise e a significância de cada eixo de maneira independente.

```
anova(cca,step=1000) # teste global

anova(cca,step=1000,by="axis") # por eixo
```

Exercite a função `ordistep` e `vif.cca` e compare com os resultados obtidos na RDA.

10.0. Estatística espacial

Atualmente a estatística espacial é um dos maiores e mais importantes campos da Ecologia, impulsionado pelo grande número de dados disponíveis. É crescente o entendimento da necessidade em se inserir modelos espaciais de forma explícita em hipóteses ecológicas. O fato de a grande maioria dos métodos empregados na investigação da distribuição espacial dos organismos parte da premissa de que nossas observações são independentes, nos obriga a investigar e testar a realidade deste pressuposto. O termo autocorrelação refere-se ao grau de correlação de uma variável com ela mesma. Desta forma, autocorrelação espacial significa que a relação entre os valores de uma variável, em locais distintos, pode ser descrita por uma função espacial. Se uma variável está autocorrelacionada a uma determinada distância x, as observações realizadas no campo de domínio desta distância não podem ser consideradas independentes. A autocorrelação é gerada por forçantes bióticas intrínsecas à comunidade (como eventos de assentamento do bentos marinho, a dispersão de larvas e propágulos de plantas ou interações bióticas

como a predação e competição). Uma dependência espacial induzida significa que o fenômeno é gerado por forçantes ambientais que, por si só, estão espacialmente estruturados (como as variáveis de um gradiente batimétrico, o gradiente de uma montanha ou um gradiente estuarino).

10.1. Correlogramas

O índice de Moran (*I*) é uma das principais e mais comuns medidas de correlação espacial e, de uma maneira geral, a álgebra empregada é semelhante à utilizada na correlação de Perason. Para se computar o índice levamos em consideração uma matriz de coordenadas em que a distância dos pontos é convertida para uma classe d. Vamos utilizar o data frame `gilgais` do pacote MASS para computarmos índice de Moran em classes de distância de um transecto regular.

```
library(MASS)

library(spdep)

data(gilgais)

grid<-as.matrix(expand.grid(1,c(seq(0,1456,4))))

nrow(grid)

plot(grid)

spacial.matrix<-dnearneigh(grid,0,4)#    criando    um
conjunto de dados de conexão em intervalos de raio
igual a 4 metros (ex. lag1=0 a 4; lag2=4 a 8...)

summary(spacial.matrix)

ph0<-gilgais$pH00

correlograma<-
sp.correlogram(spacial.matrix,ph0,order=100,method="I"
,zero.policy=T)

print(correlograma,  p.adj.method="holm")  #  corrigido
pelo método de Holm

plot(correlograma)
```

A função `mantel.correlog` testa a presença de autocorrelação multivariada utilizando o teste de Mantel.

```
cor.mantel<-
mantel.correlog(dist(gilgais),XY=grid,nperm=99,
cutoff=T)

plot(cor.mantel)

cor.mantel$mantel.res
```

10.2. Introduzindo o modelo espacial utilizando polinômios

Podemos estabelecer um modelo espacial a partir das coodenadas x-y utilizando polinômios de ordem n. A matriz contendo n vetores descreverá a estrutura espacial em questão, e pode ser utilizada em regressões múltiplas e análises canônicas. A rotina é comumente conhecida como *trend surface analysis*. Vamos utilizar o conjunto de dados da Praia Vermelha para especificarmos o polinômio que representará a matriz espacial do grid de amostragem.

```
grid<-expand.grid(1:10,1:10)

grid.c<-scale(grid,scale=F) # Para gerar polinômios,
devemos primeiramente centrar as variáveis.

poly<-poly(as.matrix(grid.c),degree=3,raw=F)

fauna.hel<-decostand(fauna,"hel")#transformação de
Hellinger

rda.gradiente<-
rda(fauna.hel~.,data=as.data.frame(poly)) # aqui
estamos testando se o gradiente espacial explica de
maneira significativa a variabilidade da fauna

RsquareAdj(rda.gradiente)$adj.r.squared)

anova.cca(rda.gradiente,step=1000) # caso o gradiente
modele de maneira significativa seus dados, as
análises posteriores para investigação de
autocorrelação devem ser efetuadas utilizando o
resíduo do modelo linear entre a fauna e o gradiente
```

Podemos testar a significância dos eixos da RDA. Os scores dos eixos que explicam de maneira significativa a ordenação dos dados podem ser utilizados em uma regressão com as variáveis abióticas.

10.3. Gerando autovetores de matrizes de vizinhos: PCNM

A variabilidade espacial pode ocorrer em diversas escalas e devemos ser capazes de gerar funções que descrevem esta variabilidade ao longo deste espectro espacial. Neste sentido, a geração de componentes principais de matrizes de vizinhos (PCNM, caso particular de mapas de Moran), que podem ser incorporadas em análises canônicas, são funções mais robustas comparadas a funções espaciais polinomiais, já que descrevem uma ampla variabilidade de escalas espaciais de maneira independente, ou seja, componentes ortogonais. De maneira resumida, o método consiste em calcular a distância Euclidiana de uma matriz de coordenadas e truncá-la pela distância mínima entre pontos. Em segundo passo, computar uma PCoA da matriz truncada e posteriormente utilizar os autovetores referentes aos autovalores positivos em regressões múltiplas ou RDA. Como os autovalores exprimem de forma decrescente a variabilidade em distintas escalas, ou seja, o primeiro autovalor e seu respectivo autovetor exprimem a maior escala de variabilidade, o segundo a segunda maior escala e assim por diante. Primariamente vamos definir as PCNM's utilizando a função `pcnm` do vegan e depois testar a autocorrelação.

```
pcnm.matrix<-pcnm(dist(grid))

pcnm.matrix$values

pcnm.matrix$vectors

pcnm.vectors<-as.data.frame(pcnm.matrix$vectors)

rda.pcnm<-rda(fauna.det~.,data= pcnm.vectors)

anova.cca(rda.pcnm) # caso significativa, teste os eixos da RDA e regrida os scores contra as variáveis abióticas
```

10.4. Incorporando modelos espaciais em análises ecológicas: particionando a variabilidade

A aplicação imediata das matrizes de funções que descrevem modelos espaciais é incorporá-las de forma explícita na hipótese ecológica a ser testada. O procedimento pode ser realizado, por exemplo, através de regressões, RDA e CCA parciais ou não. Sendo assim, em um problema envolvendo três matrizes (uma matriz resposta, a biológica, e duas explicativas, a ambiental e a espacial) a variabilidade pode ser partilhada (função `varpart` do vegan) de tal maneira que seja possível se averiguar a magnitude de explicação de cada partição e da interseção entre ambas. A análise pode ser efetuada sem a influência de um possível gradiente (*trend*), utilizando-se o resíduo da regressão entre as matrizes

biológicas e das coordenadas espaciais (um modelo polinomial por exemplo), ou o próprio gradiente pode ser inserido na análise como uma fonte de variabilidade sendo representado por uma terceira matriz.

```
anova(rda(fauna.hel,grid))  # se a relação for significativa, trabalhe com o resíduo; ou insira o gradiente como uma terceira matriz e trabalhe com os dados não regredidos

fauna.det<-resid(lm(as.matrix(fauna.hel)~.,data=grid))

pcnm<-pcnm(dist(grid),dist.ret=TRUE)  #criando as PCNM's

pcnm<-as.data.frame(pcnm$vectors)

sed.sdz<-decostand(sedimento,"standardize")

part.var<-varpart(fauna.hel,pcnm,sed.sdz)

part.var #observe o resultado e veja que você pode calcular cada R2 ajustado realizando RDA's parciais das partições indicadas em frações individuais. O resultado é exatamente o mesmo
```

Bibliografia Recomendada

Borcard, D., Gillet, F., Legendre. 2011. Numerical Ecology with R, Use R. Springer, New York, USA.

Crawley, M.J. 2007. The R Book. John Wiley & Sons Ltd., Chichester, England.

Dalgard, P. Introductory Statistics with R. 2008. Springer, New York, USA.

Landeiro, V.L. 2011. Introdução ao uso do programa R. Instituto Nacional de Pesquisas da Amazônia; Programa de Pós Graduação em Ecologia.

Legendre, P., Legendre, L. 1998. Numerical Ecology. Elsevier Science, Amsterdan, Holland.

R Development Core Team. 2007. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. http://www.R-project.org. Inúmeros manuais disponíveis.

Underwood, A.J. 1997. Experiments in Ecology: their logical design and interpretation using analysis of variance. Cambridge University Press, Cambridge, England.