Disciplina IBE 875 - Modelagem de Distribuição de Espécies

Professores: Rodrigo Tardin, Maria Lucia Lorini, Mariana Mira Vasconcellos

Roteiro – Script 1 – Introdução ao ambiente R

Neste exercício, iremos explorar o ambiente R que será usado durante toda a parte prática da disciplina. O ambiente R é um conjunto integrado de software e linguagem de programação usado para manipulação de dados, elaboração de cálculos, análises estatísticas e display gráfico. Nessa disciplina, ele será usado para:

- Manipulação e armazenamento de dados.
- Operações de cálculo, especialmente com matrizes e rasters, de forma eficiente através de funções e loops.
- Análises estatísticas através de uma coleção de pacotes desenvolvidos por diversos autores.
- Visualização dos dados através de plots e mapas.

O R pode ser instalado nesse link: https://cran.r-project.org/. E para uma melhor interface gráfica, vamos utilizar a versão grátis do programa RStudio https://www.rstudio.com/products/rstudio/download/. Existem diversos recursos online para aprender a programar no R. Como recursos livres e não pagos, eu recomendo o livro "An Introduction to R" https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf e as lições do Software Carpentry "R for reproducible scientific analysis" e "Programming with R": https://software-carpentry.org/lessons/.

No tutorial abaixo, `essa fonte será usada para os comandos que estão no script do R linha por linha`. Ainda, o jogo da velha "#" é usado para fazer comentários dentro do código (ou seja, o R irá ignorar tudo após o # até a próxima linha).

- **Usando o R**

Abra o script1.R no RStudio. Para realizar qualquer tarefa no R, você pode digitar um comando na janela do "Console" e clicar em "Enter". Vamos descobrir se o R pode calcular quanto é 2 mais 2 e algumas outras operações básicas.

```
2+2
2 + 2
```

As respostas para os dois comandos acima devem dar o resultado = 4. Logo, espaços não importam nos comandos e operações do R. Agora tente:

```
2 * 3
4 / 2
10 - 5
4 ^ 2
```

Viu? O R funciona para operações matemáticas básicas e também alguns cálculos mais complexos como veremos a seguir.

- **Objetos**

Se você digitar um comando diretamente no console (como acabamos de fazer), o R imprimirá o resultado no console. Mas, geralmente queremos armazenar o resultado de um comando para usá-lo posteriormente ou para exportá-lo. Os dados dentro do R são tratados como um objeto e armazenados por um nome que você mesmo escolhe. Dependendo da natureza e estrutura dos dados, eles são armazenados de formas diferentes como veremos a seguir.

```
X <- 2  #Objeto X é igual a 2 (<- é o mesmo que =)
X
# Lembram que esse símbolo # é para comentários?
```

O objeto X acima é um vetor numérico (=2). Existem várias classes (tipos) de objetos, dentre os mais básicos estão:

- vector (numeric)
- vector (“character”)
- vector (logical)
- factor
- matrix or array
- data.frame
- list
- functions

É muito importante entender esse conceito, pois uma determinada função vai exigir uma classe específica de objeto para funcionar. Por exemplo, se você vai fazer uma modelagem de distribuição de espécies, você vai precisar de um objeto do tipo “RasterStack” (para dados ambientais) e um objeto do tipo “data.frame” (para dados de ocorrência) para a espécie que deseja modelar. Vetores podem ser principalmente numéricos (1, 2, 3), caracteres (“a”, “B”, “temp”,”Rain”) ou lógicos (TRUE ou FALSE). Cuidado, pois para o R letras maiúsculas são diferentes das minúsculas, logo “temp” é diferente de “Temp”! Se você for somar algo ou fizer qualquer cálculo, o caracter tem que ser numérico (dados numéricos), mas se o dado for categórico (“azul”, “vermelho”) ele será armazenado como “character” sempre entre aspas.

- **Vetores numéricos**

A estrutura mais simples de objeto é o vetor numérico, que é uma entidade única que consiste em uma coleção ordenada de números. Para configurar um objeto como vetor denominado x, consistindo em cinco números, a saber 1, 2, 4, 8 e 10, vamos usar a função "c" (concatenar). Note que todas as funções (falaremos delas depois) são seguidas de ()

```
x <- c(1, 2, 4, 8, 10)
# Também pode ser do modo abaixo
c(1, 2, 4, 8, 10) -> x
# Ou ainda
x = c(1, 2, 4, 8, 10)
```

Se você digitar qualquer caracter ou uma sequência de caracteres (sem espaços) no console sem “aspas”, o R primeiro tentará interpretar isso como um objeto que você tem salvo. Se não tiver um objeto com esse nome, tentará interpretar como uma função. Se não houver nenhuma disponível, acarretará em um erro. Você pode verificar o que está dentro do objeto “x” que acabou de criar, basta digitar x (sem “”) no terminal.

```
x
# Tente agora com a letra maiúscula abaixo
X
```

Viu como os objetos x e X são diferentes? Tome muito cuidado! Agora, você pode rodar expressões matemáticas usando os objetos que criou.

```
1/x
# Lembrem-se que x é uma sequência de números, logo a expressão acima é idêntica a de baixo
1/c(1, 2, 4, 8, 10) #Números decimais sempre com .
X+1*2
# Crie objetos com outros objetos
y <- c(1.5, 0.002, 3) #Números decimais no R sempre com .
w <- c(X, 0, x)
w
z <- 1:5 #sequências consecutivas de números com :
mean(z) #Computa a média dos valores
# Vamos confirmar se o valor acima está correto
media <-(1+2+3+4+5)/5
```

```
media # o mesmo valor de mean(z)
sd(x) #Computa o desvio padrão
# Valores faltantes são representados por NA
x.missing.data <- c(x, NA)
x.missing.data
# Função para checar se há dados faltantes
is.na(x.missing.data)
# Qual elemento do objeto está faltante?
which(is.na(x.missing.data)) #funções which() e is.na()
length(x)
length(x.missing.data)
length(which(is.na(x.missing.data)))
```

- **Vetores de caracteres**

As sequências de caracteres são inseridas usando aspas duplas (“) ou simples (‘). Portanto, tudo o que estiver entre aspas será interpretado como caracter, incluindo números entre aspas (“1”).

```
labs <- c(“a”, “b”, “c”, “d”, “e”)
labs
```

A seleção de elementos nos objetos é feita com [ ]. Vamos acessar o primeiro elemento de x e o quinto elemento de labs.

```
x[1]
labs[5]
```

Podemos ainda selecionar mais de um elemento e criar um objeto ‘sublabs’:

```
sub.labs <- labs[c(2,4)]
sub.labs
```

É sempre bom nomear os elementos de um determinado objeto para melhor manipulá-lo. Então, vamos nomear x com labs:

```
x
names(x) <- labs
x
```

Viu que agora o [1] que tinha antes dos elementos sumiu? Isso porque agora os elementos não precisam ser numerados pelo R, pois eles têm nomes (a b c d e). Use a função str() para checar a estrutura de x e y:

```
str(x)
str(y)
```

Você pode acessar os elementos maiores que 5 usando uma expressão como abaixo:

```
(x[x > 5])
names(x[x > 5])
```

Você também pode remover elementos usando - . Vamos remover o quinto elemento.

```
x[-5]
x
```

Mas, veja bem. Isso só será salvo se vc criar um objeto.

```
v <- x[-5]
v
```

Vamos remover agora o primeiro e o quinto elemento de x.

```
digitos.pares <- x[-c(1,5)] #Dê nomes informativos para seus objetos (sem espaços)
digitos.pares
```

Ficou perdido e quer lembrar quantos objetos criamos até agora? Use a função ls( ).

```
ls( )
```

PERGUNTA 1: Quantos objetos foram criados até agora? Quais são eles?

- **Vetores lógicos**

Assim como em vetores numéricos, o R permite manipular expressões lógicas. O elemento de um vetor lógico pode ter os valores TRUE, FALSE, NA ("not available"). Vetores lógicos são gerados por expressões condicionais. Veja abaixo por exemplo, a condição de z ser maior que 4.

```
z
big.numbers <- z > 4
```

Os operadores lógicos são <, >, <=, >=, = = (exata igualdade), ! = (desigualdade). Ainda, se a condição1 (c1) e condição2 (c2) são expressões lógicas, `c1 & c2` é a interseção delas (c1'e'c2), `c1 | c2` é a união das suas condições (c1'ou'c2), `!c1` é a negação da condição c1 (ou seja, o contrário de c1). Vamos ver como isso funciona na prática chamando essas condições de big.numbers (> 4) e small.numbers (< 3):

```
small.numbers <- z <3
```

```
big.numbers & small.numbers #nenhum número satisfaz as duas condições ao mesmo tempo

big.numbers | small.numbers #apenas o terceiro elemento é falso, pois não satisfaz nenhuma condição

! big.numbers

# Compare !big.numbers acima com big.numbers abaixo

big.numbers
```

- **Fatores**

Fatores fornecem maneiras compactas de lidar com dados categóricos. Vamos criar um fator com as categorias “presente” e “ausente”.

```
frutos <- c(“presente”, “ausente”, “ausente”, “presente”, “presente”)

frutos
```

Verifique qual classe de objeto fator é. Esse comando `class ( )` é muito importante para saber se o objeto está no formato requerido pela função que se vai utilizar.

```
class(frutos)

frutos <- as.factor(frutos)

class(frutos)

names(frutos) <- labs

frutos
```

- **Matrizes ou arrays**

Matrizes são generalizações de vetores em duas dimensões. Primeiro vamos criar dois vetores numéricos (a e b).

```
a <- c(1, 2, 4, 8, 10)

b <- c(0.1, 0.2, 0.4, 0.8, 1)

## Agora, vamos unir esses dois vetores como colunas

m1 <- cbind(a,b)

m1

class(a)

class(m1)

dim(m1) #dimensões da matriz

nrow(m1) #número de linhas da matriz (n row)
```

```
ncol(m1) #número de colunas da matriz (n col)
# Agora, vamos unir esses dois vetores como linhas
m2 <- rbind(a,b)
m2
## Vamos colocar nomes na nossa matriz m1
colnames(m1) <- c("col1", "col2")
rownames(m1) <- labs
m1
```

Você pode acessar os elementos da matriz usando [ ], primeiro vem a linha, depois a coluna [linha, coluna]

```
m1[4,2]
# E também podemos selecionar elementos colocando os nomes das linha e colunas como abaixo
m1["d","col2"]
```

PERGUNTA 2: Qual o valor do elemento que está na coluna1 e linha 3? Que comando vc usou para descobrir?

- **Data.frame**

Os data.frames são estruturas semelhantes a matrizes, porém as colunas podem ser de diferentes tipos e não apenas numéricas. Pense em data.frames como "matrizes de dados" com uma linha por observação, mas com variáveis numéricas e (possivelmente) categóricas. Vamos criar um data.frame abaixo juntando os vetores a, b, frutos como colunas usando o comando data.frame ( )

```
plantas <- data.frame(a, b, frutos)
plantas
```

Assim como em uma matriz, você pode alterar os nomes com colnames( ) e rownames( ), primeiro vamos criar novos nomes com a função paste ( ). Se tiver dúvida em como utilizar as funções, use ?nome_da_função. Veja abaixo para sabermos mais sobre a função paste( ) que usamos para concatenar caracteres.

```
?paste #Leia as instruções na janela Help ao lado do Console
new.row.names <- paste("sp", 1:5, sep="")
new.row.names
rownames(plantas) <- new.row.names
plantas
colnames(plantas) <- c("petala", "antera", "frutos")
```

```
plantas
```

Para acessar os elementos dentro do data.frame podemos usar [ ] ou $nome.da.coluna. Vamos acessar os valores da coluna petala.

```
plantas[ ,1] #todos os elementos da coluna 1
plantas$petala
# Vamos acessar os valores de "sp3"
plantas[3,] #linha 3
plantas$petala[3] #apenas o terceiro elemento da coluna petala
```

Podemos usar o comando View( ) para visualizar toda a tabela ou data.frame.

```
View(plantas) #Pode fechar a aba depois de vizualizar
```

- **Listas**

Listas são uma forma geral de vetor em que os vários elementos não precisam ser do mesmo tipo e muitas vezes são eles próprios vetores ou listas. As listas fornecem uma maneira conveniente de retornar os resultados de uma análise estatística. Vamos fazer uma lista com nossa matriz 'm1', nosso vetor numérico 'x' e o vetor de caracteres 'labs'.

```
lista1 <- list(m1, x, labs)
lista1
```

Vamos dar nomes aos 3 elementos da lista que criamos.

```
names(lista1) <- c("m1", "x", "labs")
lista1
```

Para acessar os elementos, você pode usar [[ ]] ou $nome. Vamos acessar o terceiro elemento.

```
lista1[[3]]
lista$labs
```

Vamos acessar a primeira linha do primeiro elemento (m1).

```
lista[[1]][1,]
lista$m1[1,]
```

Removendo elementos.

```
lista[-1]
lista[-c(1,3)]
```

Você pode fazer uma lista de listas.

```
lista2 <- list(lista, lista)

lista2

lista2[[1]][[2]][3] #Acesse o terceiro elemento do segundo componente da primeira lista
```

- **Funções**

As funções são objetos do R que podem ser armazenados na área de trabalho do projeto. Isso fornece uma maneira simples e conveniente de usar o R. Você pode criar suas próprias funções ou modificar funções de outras pessoas. Os pacotes do R, disponíveis em repositórios como o CRAN (https://cran.r-project.org/) são um conjunto de funções escritas por um ou mais autores para executar uma determinada análise ou análises relacionadas. A flexibilidade do R para o desenvolvimento de pacotes para as mais diversas análises estatísticas, ou de manipulação e visualização de dados com funções executáveis em qualquer sistema operacional facilitou a reprodutibilidade da ciência e favoreceu a grande disseminação do R entre acadêmicos.

Já usamos várias funções distribuídas pelo pacote "base" que é instalado junto com o R e está sempre ativado para uso. Exemplos de funções que usamos anteriormente foram: c( ), mean( ), is.na( ), which( ), length( ), class( ), names( ), cbind( ), rbind( ), dim( ), nrow( ), ncol( ), colnames( ), rownames( ), data.frame( ), list( ). Mas, também podemos também criar nossas próprias funções. Veja abaixo.

```
## Vamos criar uma função:
f <- function() {
     print("Hello world!")
     }
## Agora vamos rodar nossa função
f()
```

Na função acima, a gente apenas pediu para imprimir a expressão "Hello world!". Mas, as funções podem incluir diversas condições e cálculos mais complexos. Não vamos nos aprofundar em como escrever funções, mas apenas em como executá-las. Existem diversas funções no pacote "base" do R e diversas outras funções em diversos outros pacotes. Vamos durante a disciplina usar essas funções já escritas por alguém.

- **Outras classes especiais**

Outras classes específicas para objetos podem ter sido criadas por quem escreveu um pacote. Mas, geralmente eles são versões "especializadas" das classes básicas acima. Por exemplo, a classe "RasterStack" é apenas um conjunto de "matrizes" com a mesma extensão, resolução e projeção espacial. Deve-se então utilizar uma função específica do pacote para criar um objeto com os dados no formato específico para utilização de tal pacote e análise. No nosso caso, precisamos instalar dentre outros, o pacote 'raster' para manipulação e preparação dos dados ambientais para análise.

- **Instalando pacotes**

Após a instalação do R e do Rstudio, você está pronto para usar as funções básicas do R (do pacote "base" que já vem pré-instalado e carregado). No entanto, normalmente você também usará "funções" de outros pacotes para uma tarefa mais específica. Por exemplo, o R não vem com uma função para lidar com dados de rasters (formato das camadas climáticas utilizadas em SDM). Se você precisar dessas funções, terá que instalar um (s) pacote (s) específico (s) para isso.

Para instalar um pacote, abra o Rstudio e vá para "Tools" → "Install Packages…" → digite o nome do pacote e instale (do repositório CRAN). Como alternativa, você pode digitar os seguintes comandos no terminal:

```
install.packages("raster", dependencies=T)
```

Com este comando, você está pedindo ao R para instalar um pacote chamado "raster" e também todas as suas dependências (ou seja, se o pacote que está instalando depende de funções de outros pacotes, eles também serão instalados). Muitos desses pacotes geram uma mensagem de 'warning' em vermelho, isso não é erro, é apenas alguma informação que o autor acha relevante você saber. Preste atenção para mensagens de erro, pois se houver, o pacote não será instalado e você tem que resolver o problema antes de instalar o pacote. Vamos agora instalar todos os outros pacotes que vamos usar na disciplina:

```
install.packages("biomod2", dependencies=T)
install.packages("rgdal", dependencies=T)
install.packages("sdmpredictors", dependencies=T)
install.packages("maptools", dependencies=T)
install.packages("usdm", dependencies=T)
install.packages("ecospat", dependencies=T)
install.packages("CoordinateCleaner", dependencies=T)
install.packages("rgbif", dependencies=T)
install.packages("spocc", dependencies=T)
install.packages("spThin", dependencies=T)
install.packages("dplyr", dependencies=T)
install.packages("dismo", dependencies=T)
install.packages("gridExtra", dependencies=T)
install.packages("tidyr", dependencies=T)
```

- **Carregando os pacotes**

Em seguida, você precisa carregar os pacotes necessários para executar suas funções. Isso é feito com o comando “library (nome_do_pacote)”. Agora, você está pronto para usar as funções disponíveis nos pacotes carregados.

```
library(raster)
library(biomod2)
library(rgdal)
library(sdmpredictors)
library(maptools)
library(usdm)
library(ecospat)
library(CoordinateCleaner)
library(rgbif)
library(spocc)
library(spThin)
library(dplyr)
library(dismo)
library(gridExtra)
library(tidyr)
```

- **Ajuda, help!**

Se souber o nome da função ou do pacote para o qual precisa encontrar instruções de como usar, você pode digitar “?nome_da_função” ou “?nome_do_pacote” ou help(função). Ainda, enquanto "?" irá procurar funções de pacotes que você carregou, “??” irá procurar funções em pacotes que você pode não ter carregado ou instalado ainda. Leia as instruções da ajuda do comando até o fim e tente rodar os exemplos que alguns disponibilizam no final para entender melhor como rodar a função.

```
?stack #veja essa função que vamos usar amanhã
help(stack) #use help( ) ou ?
?biomod2
??biomod2
```

Se não lembrar o nome do pacote, pode usar o comando abaixo e tentar procurar por palavra-chave.

```
help.start()
```

Vignettes são uma ótima maneira de aprender a usar um pacote. Alguns deles possui um vignette que são pequenos tutoriais de como rodar as funções do pacote. Se o pacote que você quer usar tem um vignette, essa informação está provavelmente dentro do

manual do pacote ou da página do pacote no CRAN. Veja aqui o pacote raster: **https://cran.r-project.org/web/packages/raster/index.html**

- **Diretório de trabalho (working directory)**

Ao rodar qualquer coisa no R, a primeira coisa a fazer é definir seu diretório de trabalho (ou seja, o diretório onde os arquivos de entrada estão localizados e onde os resultados serão salvos). Nunca comece a rodar nada no R, se você não souber em que pasta está. Para definir o diretório de trabalho no RStudio, vá para "Session" → "Set Working Directory" → "Choose Directory…". Como alternativa, você pode também digitar o seguinte comando:

```
setwd("C:/Desktop/my_folder") #Esse caminho está no formato do Windows. Se o seu computador for Mac ou Linux, o formato do caminho é diferente.

# Ou…

my.wd <- "C:Users/Documents/my_folder"

setwd(my.md)

#Tente mudar o caminho entre aspas acima "C:Users/Documents/my_folder" para o caminho da pasta da disciplina no seu computador. Se tiver dificuldade, peça ajuda aos monitores.

# Veja se funcionou usando o comando abaixo para saber em que diretório você está

getwd()
```

- **Criando um script**

É sempre uma boa ideia criar um script para qualquer análise. Para criar um script no RStudio, vá para "File" → "New file → R Script". Ou clique no botão mais a esquerda do menu (com um sinal + em verde numa folha em branco). Digite os comandos um a um em cada linha separada. Depois, você pode salvar o script para repetir a análise e para uso futuro com diferentes arquivos de entrada.

Para executar um comando a partir do script: selecione uma ou mais linhas no script e clique em "Run" (ou pressione ctrl + enter). Alternativamente, copie e cole o comando na janela "console" e pressione enter. Aqui está um exemplo de um script simples. Normalmente, ele começa carregando os pacotes necessários, definindo o diretório de trabalho e, em seguida, o código. Deixar o script bem anotado com comentários informativos também é sempre uma boa ideia. Você no futuro irá agradecer! O que parece obvio agora pode não parecer daqui a um mês quando você precisar abrir esse código novamente para re-rodar algo que um revisor pediu. Veja um exemplo de script abaixo:

```
###################################
##### R Script SDM species X  #####
###################################

library(raster)

### Mude seu diretório aqui ###
setwd("C:/My_working_directory_path")


### Pontos de ocorrência
#########################

pres <- read.csv("Ocurrence_points_data.csv")
head(pres)


### Preditores ambientais
#########################

env <- read.csv("table_evironmental_data.csv")
env <- env[,-1]
head(env) #veja as primeiras 5 linhas a tabela
```

- **Importando e exportando dados**

Dependendo da natureza dos seus dados, você precisará encontrar a função apropriada para importar / exportar seus dados para o R (geralmente o mesmo pacote fornece ambos). Por exemplo, se você deseja importar uma planilha do excel em formato csv (comma separated values), você pode usar "read.csv" ou "read.table". Se você quiser importar uma árvore filogenética você pode usar "read.phylo", para importar um alinhamento de DNA você usa "read.dna", para importar rasters ambientais, você usa o comando "raster" do pacote 'raster'. Abaixo vão alguns exemplos:

- `read.csv("mydata.csv") ## lê um arquivo csv`
- `write.csv(mydata, file="mydataexport.csv") ## exporta um arquivo csv`
- `raster("elevation.asc") ## lê um raster em formato asc`
- `readShapeSpatial("world.shp") ## lê um shapefile`

```
#Se você conseguiu mudar o caminho acima para a pasta da disciplina, tente ler os pontos da espécies Procnias nudicolis abaixo


read.csv(".Procnias/procnias_nudicollis.csv", header=T, sep=";")
```

```
#O arquivo 'procnias_nudicollis.csv'está dentro da
pasta 'Procnias'
```

Além disso, se você estiver executando um script longo e por algum motivo quiser salvar seu espaço de trabalho (ou seja, todos os objetos que você tem em sua sessão do R), você pode salvar uma imagem e carregá-la posteriormente. Para salvar, clique no pequeno disquete azul no painel superior direito ("Environment"). Você também pode usar o comando "save.image".

```
save.image(file="working.RData")
load("working.RData")
```

- **Comandos importantes para lembrar**

```
setwd( )
getwd( )
head( )
paste( )
class( )
list.files( )
```

- **Plots para visualização dos dados**

```
plot(x, type="p") #plot de pontos
plot(x, type="l") #plot de linhas
?plot #veja as instruções para o plot genérico no
pacote base e os argumentos que esse comando utiliza
```

PERGUNTA 3: O que os argumentos main, xlab, ylab da função plot( ) significam?

```
barplot(x)
boxplot(x)
hist(c(x, z, x))
hist(c(x, z, y), main = "Histograma de x, y, z", xlab
= "")
?par #veja os parâmetros gráficos que podem ser usados
nos plots
```

#FIM do Script 1