2. Primeiros Passos

Instalação do R

Os arquivos necessários para instalar o ambiente R estão disponíveis gratuitamente no sítio oficial http://www.r-project.org/.

A página oficial do R é a referência básica para seus usuários e desenvolvedores, onde você também encontra instruções de instalação, listas de discussão, tutoriais, documentação e muitas informações úteis.

Iniciando o R

Em ambiente UNIX (como o Linux, por exemplo), podemos iniciar o R a partir do interpretador de comandos (*shell*) digitando o comando R:

```
parsival@jatai $ R
```

Já no sistema MS-Windows, é necessário clicar no ícone apropriado (no desktop) ou buscar o programa a partir do menu **Iniciar**.

Independentemente de como inicia, o R apresenta uma tela com (aproximadamente) a seguinte forma:

```
R version 2.7.0 (2008-04-22)
Copyright (C) 2008 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R é um software livre e vem sem GARANTIA ALGUMA.
Você pode redistribuí-lo sob certas circunstâncias.
Digite 'license()' ou 'licence()' para detalhes de distribuição.

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

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

[Área de trabalho anterior carregada]

>
```

A Linha de Comando

O R é uma linguagem interativa, ou seja, que permite ao usuário enviar um comando por vez e receber o resultado[1)]. Para isso, usamos a linha de comando, que tem o sinal “>” quando o R está pronto para receber um comando.

Os outros dois estados da linha de comando são o de execução e o de espera para a conlusão do comando. No modo de execução não é exibido nenhum sinal e não é possível digitar outro comando. Você só perceberá isso se der um comando que tenha um tempo de execução muito longo.

O estado de espera ocorre quando o usuário envia um comando incompleto, o que é indicado por um sinal de “+”:

```
>
> log(2
+ )
[1] 0.6931472
>
```

Na primeira linha, não fechamos os parênteses da função `log` e demos *enter*. O R responde com o sinal de espera (+), indicando que o comando está incompleto. Digitando o parêntese que falta e apertando a tecla *enter* novamente o R retorna o resultado do comando, precedido de um índice numérico.[2)]

Sintaxe Básica dos Comandos

Um comando no R em geral inclui uma ou mais funções, que seguem a seguinte sintaxe:

função(argumento1 = valor, argumento2 = valor , ...)

Como nos exemplos abaixo:

```
> plot(x=area,y=riqueza)
> plot(area, riqueza)
> plot(area,riqueza,xlab="Área (ha)", ylab="Riqueza")
```

- No primeiro caso, o valor de cada argumento usado está explicitado. O argumento x da função `plot` é a variável independente, e o argumento y é a variável dependente.
- Se o nome dos argumentos é omitido, como no segundo caso, o R usa o critério de posição: o primeiro valor é atribuído ao primeiro argumento, o segundo valor ao segundo argumento, e assim por diante. Como os dois primeiros argumentos da função plot são x e y, o segundo comando acima equivale ao primeiro.
- Os dois critérios podem ser combinados, como no terceiro comando: como x e y são os dois primeiros argumentos, não é preciso declará-los. Como os outros dois argumentos que se deseja usar (`xlab` e `ylab`) não vêm em seguida, é preciso declará-los.

Mais detalhes na seção sobre funções.

Criação de Objetos: Atribuição

Você pode “guardar” o resultado de um comando com a operação de *atribuição*, que tem a sintaxe:

objeto *recebe* **valor**

Há dois operadores que atribuem valores a um objeto dessa maneira:

- Sinal de menor seguido de hífen, formando uma seta para a esquerda: <-
- Sinal de igual: =

Uma forma de atribuição menos usada é:

valor *atribuído a* **objeto**

Nesse caso, o sinal de atribuição é o hífen seguido de sinal de maior, formando uma seta para direita: ->

Há, portanto, três maneiras de guardar os resultados de um comando em um objeto:

```
> a <- sqrt(4)
> b = sqrt(4)
> sqrt(4) -> c
```

Para exibir o conteúdo de um objeto, basta digitar seu nome

```
> a
[1] 2
> b
[1] 2
> c
[1] 2
```

Se a atribuição é para um objeto que não existe, esse objeto é criado. **Mas cuidado**: se já há um objeto com o mesmo nome na sua área de trabalho, seus valores serão substituídos:

```
> a <- sqrt(4)
> a
[1] 2
> a <- 10^2
> a
[1] 100
```

Mensagens de Erro e de Avisos

Como em qualquer linguagem, o R tem regras de sintaxe e grafia. Mas contrário das linguagens humanas, mesmo um pequeno erro torna a mensagem incompreensível para o R, que então retorna uma mensagem de erro:

```
> logaritmo(2)
Erro: não foi posssível encontrar a função "logaritmo"
> log(2))
Erro: unexpected ')' in "log(2))"
> log(2, basse=10)
Erro: unused argument(s) (basse = 10)
> log(2, base=10)
[1] 0.30103
```

Em outros casos, o comando pode ser executado, mas com um resultado que possivelmente você não desejava. O R cria mensagens de alerta para os casos mais comuns desses resultados que merecem atenção :

```
> log(-2)
[1] NaN
Warning message:
In log(-2) : NaNs produzidos
```

Para Sair

Para sair do R, a forma mais fácil é usar o comando q (do inglês *quit*). Nesse caso o R, lhe pergunda se você deseja *salvar* (gravar) sua sessão de trabalho.

```
> q()
Save workspace image? [y/n/c]:
```

As opções são: `y = yes, n = no, c = cancel`.

Como o R Guarda os Dados?

O que significa a pergunta feita quando damos o comando `q()`?

A resposta passa por três conceitos importantíssimos, que são o **diretório de trabalho**, a **sessão** e a **área virtual de trabalho (*workspace*)**.

Cada vez que você inicia o R, dizemos que se inicia uma **sessão**.

O diretório a partir do qual você iniciou o R é o **diretório de trabalho** dessa sessão. Para verificar seu diretório de trabalho, use o comando `getwd`[3]:

```
> getwd()
[1] "/home/paulo/work/Pos_grad/Eco_USP/cursoR"
```

Para alterar o diretório de trabalho há a função `setwd`:

```
> setwd("/home/paulo/work/treinos_R/")
> getwd()
```

```
[1] "/home/paulo/work/treinos_R"
```

A sessão iniciada está ligada a uma área de trabalho particular chamada de **workspace**.

Tudo o que você faz durante uma sessão (leitura de dados, cálculos, análises estatísticas) é realizado no **workspace**.

Mas o **workspace permanece na memória do computador**. Somente quando você dá o comando de sair (`q()`) é que o R lhe pergunta se você deseja **gravar** o seu **workspace**. Se você responde 'y', o R grava um arquivo chamado `.RData`[4] em seu **diretório de trabalho**. Na próxima vez que o R for chamado desse diretório, o conteúdo do arquivo `.RData` será carregado para o "workspace".

Aprenda este Comando para Não Perder Seu Trabalho

Se acontecer do computador ser desligado durante uma sessão do R, tudo que foi feito será perdido!!! Para evitar isso, é interessante gravar com frequência o **workspace** utilizando o comando '`save.image()`':

```
> save.image()
>
```

Por *default*, o R gravará o workspace no arquivo `.RData`, e quando você reiniciar uma sessão, o R automaticamente **carrega** esse arquivo. Mas você pode salvar em outro arquivo utilizando o **argumento** `file` da função:

```
> save.image(file="minha-sessao-introdutoria.RData")
>
```

Como o R carrega automaticamente apenas o arquivo `.RData` que está no diretório de trabalho, caso deseje carregar outros arquivos você deverá usar a função `load`:

```
># Carrega um arquivo de workspace no mesmo diretório
> load(file="minha-sessao-introdutoria.RData")
># Carrega o arquivo default de workspace de outro diretório:
> load(file="/home/paulo/work/treinos_R/.RData")
```

No código acima podemos ver o símbolo "#" seguido de comentários que explicam o que a função está fazendo. Quando você coloca "#", o R irá ignorar o que vem depois do símbolo na linha. Ou seja, se você copiar uma linha começando com "#" no console e pedir para o R executar a mesma, não vai acontecer nada. Este símbolo é muito útil para comentar o seu código.

Se você quiser salvar apenas alguns objetos (digamos, os resultados das suas análises), você pode usar o comando `save`:

```
> save(modelo1, file="meu_modelo.RData")
> save(dados, modelo1, modelo2, file="meus_modelos.RData")
```

Tome cuidado com a sintaxe do comando `save`: ele aceita o nome de vários objetos que existem no seu workspace e um nome de arquivo, que deve ser passado com `file=`.

O comando save aceita o resultado de outros comandos. Por exemplo, o código abaixo equivale ao comando save.image():

```
> save (list=ls(), file="tudo.RData")
```

Aprenda este Procedimento para Organizar Seu Trabalho

Crie um diretório de trabalho para cada análise, ou mesmo para versões diferentes da mesma análise, e chame o R desse diretório. Fazendo isso você recupera seu trabalho de maneira simples, bastando salvar as alterações regularmente com o comando `save.image()` e confirmar a gravação das alterações ao encerrar a sessão.

Ao contrário do que você pode estar acostumado(a), não é uma boa idéia manter vários arquivos com diferentes versões dos dados ou análises em um mesmo diretório. Os usuários de R em geral mantém o padrão da linguagem, de um único arquivo *default* por análise, o `.RData`, criando quantos diretórios forem necessários para organizar o trabalho.

Gerenciando a Área de Trabalho

Listando Objetos

O comando `ls` lista todo o conteúdo da área de trabalho, se não é fornecido argumento:

```
> ls()
[1] "consoantes" "CONSOANTES" "vogais" "VOGAIS"
```

A função `ls` possui argumentos que podem refinar seus resultados, consulte a ajuda para os detalhes.

Apagando Objetos

O comando `rm` apaga objetos da área de trabalho:

```
> ls()
[1] "consoantes" "CONSOANTES" "vogais"     "VOGAIS"
> rm(consoantes)
> ls()
[1] "CONSOANTES" "vogais"     "VOGAIS"
```

Consulte a ajuda da função `rm` para seus argumentos.

Exercícios

2.1. Exercício para Usuários Windows: *Diretório de Trabalho*

1. Crie um diretório para seus exercícios da disciplina.
2. Chame o R, clicando no ícone da área de trabalho ou na barra de tarefas.
3. Verifique o seu diretório de trabalho.
4. Mude o diretório de trabalho para o diretório que você criou.
5. Verifique o conteúdo da área de trabalho.
6. Carregue o arquivo letras.RData (apagar extensão .pdf).
7. Verifique novamente sua área de trabalho.
8. Saia do R, tomando o cuidado de salvar sua área de trabalho.
9. Repita os passos 2 a 5.

Pergunta: Que problemas você percebeu? Há uma maneira de iniciar o R no windows que evite esses problemas?

Como Conseguir Ajuda no R

No R é essencial aprender a procurar auxílio e manuais sozinho. Após um aprendizado inicial, não há meio de evoluir no conhecimento do ambiente R se não se buscar os ***helps*** que ele possui.

O Comando ''help''

Para conseguir ajuda sobre um comando pode ser usada a função `help`:

```
> help(mean)
```

ou então o operador de ajuda que é o sinal de interrogação '?':

```
?mean
```

No caso se buscar ajuda sobre um **operador** (símbolo para operações aritméticas e algébricas) devemos utilizar as aspas duplas:

```
> help("+")
> ?"*"
```

Ao utilizar esses comandos (help e ?) o R abre uma página hipertexto no seu navegador, contendo as informações de ajuda para o tema solicitado (função ou operador).

Help no terminal ou linha de comando do prompt

Caso esteja rodando o R diretamente no shell ou no prompt do windows, ao rodar a função `help` ele entrará no modo ***help interativo***.

Essa ***help page*** pode ser examinada usando os seguintes comandos de teclado:

- tecla de espaço = um página para frente;
- tecla “f” = uma página para frente;
- tecla “b” = uma página para trás;
- tecla enter = uma linha para frente;
- tecla j = uma linha para frente;
- tecla k = uma linha para trás;
- tecla q = sai do modo **help interativo** e retorna à linha de comando.

No modo **help interativo** também é possível se fazer uma busca por uma palavra (*string*) usando a tecla '/', na forma:

```
/string
```

Executado um comando de busca temos dois novos comandos válidos:

- tecla n = próxima string, e
- tecla N = string anterior.

A Tela de Ajuda

O conteúdo do **help** pode parecer árido à primeira vista, mas é extremamente informativo, assim que nos acostumamos com ele. As seções são padronizadas, e seguem sempre a mesma ordem:

- Um cabeçalho com o nome da função, o pacote do R à qual pertence, e a classe do documento de ajuda
- O nome completo da função
- A sintaxe da função, que pode estar especificada para diferentes tipos de dados ou métodos
- A explicação de cada um dos argumentos da função
- O valor retornado pela função
- Referências bibliográficas
- Indicação de outras funções relacionadas
- Exemplos (colar esses comandos no R é uma das melhores maneiras de entender uma função).

Abaixo o conteúdo da ajuda para a função ''mean'':

```
mean                   package:base                   R Documentation

Arithmetic Mean

Description:

     Generic function for the (trimmed) arithmetic mean.

Usage:

     mean(x, ...)
```

```
     ## Default S3 method:
     mean(x, trim = 0, na.rm = FALSE, ...)

Arguments:

       x: An R object.  Currently there are methods for numeric data
          frames, numeric vectors and dates.  A complex vector is
          allowed for 'trim = 0', only.

    trim: the fraction (0 to 0.5) of observations to be trimmed from
          each end of 'x' before the mean is computed.

   na.rm: a logical value indicating whether 'NA' values should be
          stripped before the computation proceeds.

     ...: further arguments passed to or from other methods.

Value:

     For a data frame, a named vector with the appropriate method being
     applied column by column.

     If 'trim' is zero (the default), the arithmetic mean of the values
     in 'x' is computed, as a numeric or complex vector of length one.
     If any argument is not logical (coerced to numeric), integer,
     numeric or complex, 'NA' is returned, with a warning.

     If 'trim' is non-zero, a symmetrically trimmed mean is computed
     with a fraction of 'trim' observations deleted from each end
     before the mean is computed.

References:

     Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) _The New S
     Language_. Wadsworth & Brooks/Cole.

See Also:

     'weighted.mean', 'mean.POSIXct'

Examples:

     x <- c(0:10, 50)
     xm <- mean(x)
     c(xm, mean(x, trim = 0.10))

     mean(USArrests, trim = 0.2)
```

Ajuda em Hipertexto: ''help.start''

Para os que preferem trabalhar com ajuda em hipertexto (html), há o comando `help.start()`:

```
> help.start()
Making links in per-session dir ...
If 'sensible-browser' is already running, it is *not* restarted, and
    you must switch to its window.
Otherwise, be patient ...
>
```

Após esta mensagem, um portal de ajuda em hipertexto será aberto no navegador de internet padrão do sistema. A partir dessa página inicial é possível fazer pesquisas por palavras-chave, consultar funções por pacote ou por ordem alfabética, obter textos introdutórios sobre a linguagem, e muito mais.

Pesquisa por Palavras-Chave na Ajuda

Outro comando muito útil é o `apropos`. Ele possibilita sabermos quais funções do R tem no nome uma certa palavra (*string*):

```
> apropos(plot)
 [1] "biplot"              "interaction.plot"    "lag.plot"
 [4] "monthplot"           "plot.density"        "plot.ecdf"
 [7] "plot.lm"             "plot.mlm"            "plot.spec"
[10] "plot.spec.coherency" "plot.spec.phase"     "plot.stepfun"
[13] "plot.ts"             "plot.TukeyHSD"       "preplot"
[16] "qqplot"              "screeplot"           "termplot"
[19] "ts.plot"             "assocplot"           "barplot"
[22] "barplot.default"     "boxplot"             "boxplot.default"
[25] "cdplot"              "coplot"              "fourfoldplot"
[28] "matplot"             "mosaicplot"          "plot"
[31] "plot.default"        "plot.design"         "plot.new"
[34] "plot.window"         "plot.xy"             "spineplot"
[37] "sunflowerplot"       "boxplot.stats"       ".__C__recordedplot"
```

Para pesquisas mais complexas e refinadas há ainda a função `help.search()`. Por exemplo, para pesquisar funções que tenham a palavra “skew”no título:

```
> help.search(field="title","skew")

Help files with title matching 'skew' using regular expression
```

```
matching:



skewnormal1(VGAM)       Univariate Skew-Normal Distribution Family
                        Function
snorm(VGAM)             Skew-Normal Distribution
k3.linear(boot)         Linear Skewness Estimate



Type 'help(FOO, package = PKG)' to inspect entry 'FOO(PKG) TITLE'.
```

Busca de Exemplos e de Argumentos das Funções

Para obter apenas os exemplos que estão na ajuda de uma função, use a função `example`:

```
> example(mean)

mean> x <- c(0:10, 50)

mean> xm <- mean(x)

mean> c(xm, mean(x, trim = 0.1))
[1] 8.75 5.50

mean> mean(USArrests, trim = 0.2)
  Murder  Assault UrbanPop     notGood
    7.42   167.60    66.20    20.16
```

Muitas vezes precisamos apenas nos lembrar dos argumentos de uma função. Para isso, use a função `args`, que retorna os argumentos de uma função:

```
> args(chisq.test)
function (x, y = NULL, correct = TRUE, p = rep(1/length(x), length(x)),
    rescale.p = FALSE, simulate.p.value = FALSE, B = 2000)
NULL
```

Argumentos com valores atribuídos são os valores *default* da função. Por exemplo, por *default* a função de teste de Qui-quadrado estima a significância pela distribuição de Qui-quadrado e não por randomização (argumento `simulate.p.value=FALSE`).

Exercícios

Exercício 2.2. *Use a Ajuda para Conhecer Argumentos das Funções*

1. Execute o R, usando o diretório de trabalho criado no exercício anterior.
2. Use a função `load` para carregar o arquivo bichos.RData (apagar

extensão .pdf) no *workspace*.
3. Consulte a ajuda da funções `rm` e `ls` para descobrir como apagar apenas os objetos cujos nomes começam com “temp”.

Pacotes

Pacotes são conjuntos de funcionalidades (funções e dados) distribuídos em conjunto para realizar tarefas específicas. Por exemplo, o pacote **vegan** carrega na sua área de trabalho (deixa disponível para uso) um conjunto de ferramentas para análises de dados de ecologia de comunidades. Para usar os pacotes disponíveis no R [5)]

é necessário entender as diferenças entre **baixar** (download) o pacote do repositório e **carregar** em sua área de trabalho. Para baixar algum pacote disponível no repositório CRAN do R é necessário utilizar o comando *install.packages()* com o nome do pacote entre “” dentro do parenteses[6)].

```
install.packages("vegan")
```

Outra forma é usar o menu da interface gráfica e selecionar. Siga as instruções: (1) selecione o repositório mais próximo (p.ex: *Brazil(SP1)*) e em seguida navegue na barra de pacotes e selecione o que deseja. Se não houver nenhuma mensagem de erro, significa que o download do pacote foi realizado com sucesso.

Caso o pacote esteja instalado ele aparecerá entre os hiperlinks da página de ajuda hipertexto da função *help.start()*. Entre na página do pacote e navegue pelas opções e funções que forem de seu interesse. Escolha uma função (decorana) e em seguida tente apresentar o ajuda dela pelo R:

```
help.start()
help(decorana)
```

A mensagem (ou algo similar): *“No documentation for ‘decorana’ in specified packages and libraries...”*, significa que a sua sessão do R não encontrou a documentação referente a função, apesar do pacote estar instalado em nosso computador. Isso aconteceu porque não carregamos a função em nossa área de trabalho, para cada projeto, precisamos carregar aqueles pacotes que vamos necessitar (normalmente nas primeiras linhas de comando do nosso código):

```
library(vegan)

example(vegan)
```

Podemos imaginar a nossa sessão do R como uma bancada de trabalho em uma oficina, cercada por vários armários que contém as ferramentas que precisamos para realizar uma tarefa. Dependendo da tarefa que vamos realizar (arrumar uma moto, construir uma cadeira...) abrimos os armários que contem as ferramentas necessárias à tarefa desejada e apenas esses (função *library()*). Caso não tenhamos as ferramentas necessárias para uma tarefa específica (consertar um relógio), precisamos ir na loja de ferramentas (repositório) e comprar conjunto de ferramentas de relojoeiro (função *install.packates(“watch”)* que vem em um armário que colocamos ao lado dos outros em nossa oficina.

Particularidades dos Comandos no R

Como em toda linguagem, o R tem algumas regras básicas de sintaxe e grafia:

1. O nome de comandos e objetos no R pode ser compostos de:
    - letras (minúsculas ou maiúsculas),
    - números, e
    - o ponto (.).
2. Evite qualquer outro caracter especial, incluindo *o espaço em branco*.
3. O nome *não pode ser iniciado* por um número.
4. O R diferencia maiúscula e minúsculas: o comando q é diferente do comando Q (que não existe!!!).

Para que um comando seja **executado** pelo R é necessário ser acompanhado de parênteses '()'. Compare esse comando

```
> q()
Save workspace image? [y/n/c]: c
>
```

com o comando

```
> q
function (save = "default", status = 0, runLast = TRUE)
.Internal(quit(save, status, runLast))
<environment: namespace:base>
>
```

O comando sem os parênteses é na verdade o **nome do comando**. Sendo o R um software de código aberto, toda vez que se digita o nome de um comando, ele **não executa** o comando mas **mostra o conteúdo** do comando (o código).

1) é possível também de executar um lote de comandos, mas neste wiki trabalharemos apenas com o modo interativo.

2) o significado desse índice ficará claro na seção sobre indexação. Por enquanto basta saber que os resultados do R são precedidos por um indicador numérico entre colchetes

3) acrônimo de “get working directory”

4) este é um arquivo oculto, pois seu nome inicia-se com um ponto

5) “ Currently, the CRAN package repository features 14270 available packages.” — *Alexandre Adalardo de Oliveira 2019/05/22 14:14* “Currently, the CRAN package repository features 5217 available packages.” — *Alexandre Adalardo de Oliveira 2014/02/17 14:31*

6) a princípio todas as palavras que escrevemos sem aspas no R ele busca como sendo objetos presentes em nossa área de trabalho ou pacotes carregados ou instalados