LINGUAGEM R: INTRODUÇÃO À PROGRAMAÇÃO E AUTOMAÇÃO

Umberto Mignozzetti

FGV RI

Como citar este material:

MIGNOZZETTI, Umberto. **Linguagem R: introdução à programação e automação**. Rio de Janeiro: FGV, 2022.

SUMÁRIO

PROGRAMAÇÃO E AUTOMAÇÃO EM R

Neste curso, vamos estudar um pouco do poder dos computadores, que é a capacidade de automatizar tarefas repetitivas. Vamos aprender os principais comandos para automatizar as tarefas e veremos como aplicá-los em contextos de pesquisa.

Programação em R

Computador

Uma vantagem de usar um computador é que ele permite tornar tarefas que realizamos com vários comandos as mais automatizadas possíveis. O computador, de um lado, não é capaz de nada muito inteligente, como entender uma piada ou ironia. No entanto, os computadores são ótimos em executar tarefas repetitivas, várias vezes, sem perder o nível de precisão. Desse modo, aprender a programar é, basicamente, aprender fazer o computador repetir decisões que você teria de tomar, sem que você interfira no processo.

Computador e automatização de tarefas

Suponha que temos de fazer as seguintes tarefas: receber o nome e a idade de uma pessoa, colocar a pessoa em um banco de dados e, se a pessoa tiver acima de 18 anos, vamos dizer que ela pode votar (caso contrário, que não pode). Esse tipo de tarefa repetitiva e maçante é extremamente difícil de realizarmos na vida real. No entanto, no computador, isso se torna extremamente simples: daí a primeira informação importante que você deve saber sobre computadores é que *computadores são ótimos para fazer tarefas repetitivas.* Isso porque eles conseguem fazer milhões de comparações de modo preciso e em pouquíssimo tempo. No entanto, existe algo fundamental que define se podemos ou não fazer no computador, que é a capacidade de reproduzir em pequenas e simples instruções, algo complexo e repetitivo.

Se conseguimos escrever um passo a passo não ambíguo, que chamamos de pseudocódigo, é porque também é possível um programa de computador para executar esse passo a passo. Na sequência, executar o passo a passo uma ou dez mil vezes é extremamente simples. A seguir, vejamos os comandos para transformarmos os passos em códigos de R.

Comparações

O comando `if` **se** permite que você teste uma condição no R. O uso do programa segue a seguinte lógica:

```
if (condicao_for_satisfeita) {
  comando
}
```

Ainda, podemos usar condições do tipo **se-então**, que testa uma condição (e executa uma tarefa se a condição for satisfeita) e, se ela não for satisfeita, executa uma tarefa alternativa:

```
if (condicao_for_satisfeita) {
  tarefa
} else {
  tarefa_alternativa
}
```

Se o comando tiver mais de uma condição que precise de teste, podemos usar a seguinte sintaxe:

```
if (condicao_1_for_satisfeita) {
  tarefa
} else if (condicao_2_for_satisfeita) {
  tarefa_para_condicao_2
} else if (condicao_3_for_satisfeita) {
  tarefa_para_condicao_3
} else {
  tarefa_alternativa_as_condicoes_123
}
```

Como vimos, precisamos utilizar os comparadores lógicos. Por exemplo, podemos testar se `x`:

```
x > 10              # Falso
## [1] FALSE
x <= 2              # Verdadeiro
## [1] TRUE
x == 3              # Falso
## [1] FALSE
x != 'casa'         # Verdadeiro
## [1] TRUE
x > 0 & x < 4       # x entre 0 e 4 (verdadeiro)
## [1] TRUE
!(x > 0 & x < 4)   # x não está entre 0 e 4 (ou seja, a negação da verdade
é??)
## [1] FALSE
x < 2 | x > 2       # x menor que dois ou maior que 2...
## [1] FALSE
```

Além disso, podemos testar o tipo de objeto ou valor que estamos trabalhando:

Tabela 1 – Tipos de objetos

operador	significado
is.numeric()	testa se é um número ou um vetor numérico
is.character()	testa se é um caractere ou um vetor de caracteres
is.logical()	testa se é um valor booleano
is.na()	testa se é *missing*
is.null()	testa se é resultado nulo
is.factor()	testa se uma variável é um fator
is.nan()	testa se é nan, por exemplo, raiz quadrada de um número negativo
is.finite()	testa se o número é finito, por exemplo, fruto de divisão por zero

Por exemplo:

```
is.numeric(10)      # Verdadeiro
## [1] TRUE
is.character('a')   # Verdadeiro
## [1] TRUE
is.logical('bozo') # Falso
## [1] FALSE
is.null(NULL)       # Verdadeiro
## [1] TRUE
is.finite(log(0))   # Falso
## [1] FALSE
is.na(x)            # Falso
## [1] FALSE
```

Alimentando os comparadores, podemos criar condições:

```
x <- 10
if (x > 5) {
  cat('x é um número maior que 5\n')
}
## x é um número maior que 5
```

Ou, até mesmo, condicionais mais sofisticados:

```
x <- -5
if (x > 5) {
  cat('x é um número maior que 5\n')
} else if (x>=0 & x <5) {
  cat('x está entre 0 (inclusive) e 5\n')
} else if (x==5) {
  cat('x é exatamente igual a 5\n')
} else {
  cat('x é negativo\n')
}
## x é negativo
```

Basta variar os valores de `x` para acionar cada uma das condições. Note que as condições são mutuamente excludentes, para evitar problemas de execução paralela no código.

Loops

A base da computação está na repetição de tarefas de maneira automática. Para isso, temos os *loops*.

a) *while*

O while executa um grupo de tarefas enquanto uma condição for verdadeira. Por exemplo:

```
x <- 1
while (x < 100) {
  cat(x, ', ')
  x<-x+1
}
```

b) *for*

O for é como o *while*, mas ele considera um conjunto de iteradores para fazer a repetição.

```
for (i in 1:10) {
  print(i)
}
```

O que toma a sequência que vai de 1 até 10 e imprime número por número. Podemos passar qualquer vetor para fazermos o iterador do *loop*. No caso, nomes:

```
lista_pessoas <- c("fasolin", "guisela", "gabi", "natalia")
for (i in lista_pessoas) {
  print(i)
  if (i=='natalia') {
    cat('É do time do Umberto!')
  }
}
## [1] "fasolin"
## [1] "guisela"
## [1] "gabi"
## [1] "natalia"
## É do time do Umberto!
```

Podemos, ainda, colocar um *loop* dentro de outro:

```
for (i in 1:9) {
  cat('Tabuada do ', i , '\n')
  for (j in 1:10) {
    cat (i, ' x ', j , ' = ', i*j, '\n')
  }
  cat ('---------------\n')
}
```

Podemos fazer muito mais com um *loop* dentro de outro.

Criação de funções

Muitas vezes, acabamos repetindo partes de funções várias vezes no R. Nesses casos, para evitar copiar e colar códigos, e fazermos nossos códigos elegantes e compactos, precisamos criar o que chamamos de *funções*. Para criar funções, fazemos o seguinte:

```
nome_da_funcao <- function(parametros) {
  operações
  return(valores_retornados)
}
```

Por exemplo, sabemos que todo ano bisexto é o ano em que, se dividirmos os anos por 4, a divisão é exata (ou seja, resto é zero). Nesse contexto:

```
eh_bisexto <- function(ano) {
  resto = ano %% 4
  if (resto == 0) {
    return('Ano bisexto!')
  } else {
    return('Não é ano bisexto...')
  }
}

eh_bisexto(2013)
## [1] "Não é ano bisexto..."
```

Coloquei o ano de meu nascimento, você pode colocar o do seu e testar. Note que, se eu tivesse de calcular se um ano é bisexto em diversas partes do código, bastaria trocar pelo chamado da função.

Leitura sugerida

- Wickham e Grolemund (2017), capítulo 15.
- Maindonald e Braun (2006), capítulo 1 (seção 1.4).

Vetores e matrizes em R

Operações vetoriais

Quando operamos com números, um a um, fazemos o computador ter de varrer as matrizes ou vetores de dados, e isso toma muito tempo. O R facilita algumas operações usando vetorização. Vetorização significa que a soma de dois vetores soma os valores um a um, sem precisar criar um *loop* que faça a soma valor por valor. No caso:

```
x = c(1,2,3,5,6,10)
y = c(2,10,9,5,2,1)
z <- numeric()
for (i in 1:length(x)) {
  z[i] = x[i] + y[i]
}
z
## [1]  3 12 12 10  8 11
```

A operação vetorizada:

```
x = c(1,2,3,5,6,10)
y = c(2,10,9,5,2,1)
z = x+y
z
## [1]  3 12 12 10  8 11
```

As operações vetorizadas comuns são:

Tabela 2 – Operações vetorizadas

nome	operação
2x	multiplica as entradas pelo escalar 2
x*x	multiplica as entradas uma a uma
x+y	soma as entradas uma a uma
x/y	divide as entradas uma a uma
x+2	soma cada entrada pelo escalar 2

Operações com matrizes

Temos também operações matriciais, nas quais fazemos algo similar com operações com as matrizes que vocês aprenderam no Ensino Médio. Nesse caso, considere que A seja uma matriz m x n, B seja uma matriz n x q, C uma matriz n x n, e x um vetor n x 1:

Tabela 3 – Operações com matrizes

nome	operação
**A%*%*B*	multiplica as matrizes A por B
t(A)	transpõe a matriz A
t(x)	transpõe o vetor x
A%*%x	multiplica a matriz A pelo vetor x
A*2	multiplica a matriz A pelo escalar 2

Por exemplo, suponha que `X` seja um vetor de dados, no caso variáveis independentes e uma coluna de 1's (constante) de dimensão `n x k` e `y` um vetor de resultados (variável dependente) de tamanho `n x 1`. A fórmula de mínimos quadrados ordinários, em versão matricial, fica:

```
solve(t(X) %*% X) %*% t(X) %*% y
```

Isso tem de dar resultado semelhante ao da regressão linear.

Medição do tempo de operações no R

Suponha que temos dois vetores de dados, `x` e `y`, e queremos medir o tempo de execução de duas funções:

```
# Soma entradas usando for
soma_for <- function(x,y) {
  z <- numeric()
  for (i in 1:length(x)) {
    z[i] = x[i] + y[i]
  }
  return(z)
}

# Soma vetorizada
soma_vetor <- function(x,y) {
  z <- x+y
  return(z)
}
```

Usamos a função `system.time` para isso:

```
# Soma entradas usando for
x <- 1:10000
y <- 10001:20000

# Soma com for
system.time({
  soma_for(x,y)
})
##    user  system elapsed
##   0.008   0.000   0.008
# Soma vetorizada
system.time({
  soma_vetor(x,y)
})
##    user  system elapsed
##       0       0       0
```

No *elapsed*, temos a resposta: o tempo de soma usando *for* é sempre maior do que o tempo de soma usando a operação vetorizada. Esse tipo de função é ótimo para avaliar o tempo de operações e determinar qual o melhor código para uma tarefa.

O que mais fazer com o R?

R e mapas

O R é um programa eficaz na produção de mapas e conteúdos cartográficos. Ele conta com uma série de pacotes que realizam as mais diversas operações com mapas. Uma série de exemplos de mapas pode ser encontrada no *site* de Eric Anderson (http://eriqande.github.io/rep-res-web/lectures/making-maps-with-R.html).

R e gráficos interativos

O R possui um pacote chamado Shiny. Esse pacote facilita a criação de gráficos interativos, nos quais o usário pode alterar os padrões dos gráficos e dos resultados que são apresentados. Uma videoaula da equipe do R-Studio de como construir gráficos interativos segue no link (https://www.rstudio.com/resources/webinars/interactive-graphics-with-shiny).

R e *machine learning*

A computação moderna foi praticamente revolucionada nos últimos anos pelos procedimentos de *machine learning*. A ideia central do *machine learning* é usar o computador para entender os padrões que são recorrentes nos conjuntos de dados que queremos analisar. O *webbook* de Laurent Gatto explica melhor as técnicas de *machine learning* que são de acesso fácil para o conhecedor de R: http://bit.ly/intromlr

R e *deep learning*

Entre as diversas técnicas de *machine learning*, uma técnica se destaca. O *deep learning* usa repetidas interações das técnicas de *machine learning*, visando aprofundar o conhecimento que a máquina consegue extrair dos dados analisados. Apesar dos principais modelos de *machine learning* serem estimados usando linguagens de programação, como Python, atualmente, o R está equipado para fazer grande parte das análises e estimações de *deep learning*, usando o pacote keras. Choillet e Alairre descrevem, em seu livro *Deep Learning with R*, os principais modelos que podemos ajustar usando keras.

R e *Big Data*

Nos dias de hoje, o trabalho comum dos cientistas de dados envolve o processamento e a análise de grandes volumes de dados. Em casos frequentes, os dados são tão intensivos que não cabem em uma só máquina. Chamamos esses dados de *big data*. O R tem pacotes que permitem lidarmos com esses dados de maneira eficiente e rápida. A seguir, no *link* sugerimos um *webinar* da equipe do R Studio, ensinando Big Data analytics para R: https://www.rstudio.com/resources/webinars/working-with-big-data-in-r.

R e experimentos

As ciências sociais modernas, assim como a medicina no decorrer do século XX, foram profundamente revolucionadas pelo uso de técnicas experimentais de análise. Os experimentos permitem que isolemos, de maneira eficiente, a causalidade na relação entre variáveis. O R permite a análise e montagem de experimentos de maneira simples e fácil. No *site* do EGAP, temos uma calculadora para medirmos qual a amostra necessária para detectarmos diferenças estatísticas em dados experimentais: https://egap.shinyapps.io/power-app/.

Fim

Essa apostila termina aqui. No entanto, como você viu, o R é um dos principais *softwares* de análise de dados e seu uso é extensivo, cobrindo diversas áreas da estatística, e análise e processamento de dados. O custo inicial foi alto: este curso foi difícil, mas a recompensa é aprender um *software* que é, ao mesmo tempo, ótimo para análise de dados e plástico para todas as outras operações complementares à análise de dados. Boa sorte com a sua jornada de aprofundar os conhecimentos em R!

Saiba mais: como são feitos pacotes em R

Os pacotes em R, apesar da aparência de sofisticados, na verdade, são feitos de códigos como esses que aprendemos aqui. Para criar um pacote, basta ter resolvido um problema para o qual você sabe não existir nenhum outro pacote para resolver. Nesse caso, você pode consultar o CRAN do R, para ver se existe algum pacote que resolva o problema que você teve.

Para criar um pacote, você também deve atentar para que as funções previnam dar problemas quando receberem valores inválidos. Isso pode ser facilmente corrigido avisando o R que o valor é inválido e sugerindo, no interior da função, quais valores são verdadeiros para o caso em mãos.

Por fim, você terá de documentar as suas funções, explicando para que elas servem e como podem contribuir com o trabalho da pessoa que está usando. Um bom passo é sugerir o problema que você teve e como você resolveu com essa função que você fez disponível.

BIBLIOGRAFIA

JAMES, Gareth, *et al*. *An introduction to statistical learning*. V. 112. New York: springer, 2013.

MAINDONALD, John & BRAUN, John. *Data analysis and graphics using R:* an example-based approach. V. 10. Cambridge University Press, 2006.

REYES, Jose Manuel Magallanes. *Introduction to Data Science for social and policy research*. Cambridge University Press, 2017.

TEAM, R. Core. *R language definition*. Vienna, Austria: R foundation for statistical computing (2000).

WICKHAM, Hadley & GROLEMUND, Garrett. *R for data science:* import, tidy, transform, visualize, and model data. O'Reilly Media, Inc., 2016.

WICKHAM, Hadley & FRANCOIS, Romain. *dplyr: a grammar of data manipulation*, 2013. Disponível em: https://github.com/hadley/dplyr.version 0.1.[p 1]. Acesso em: 2017.

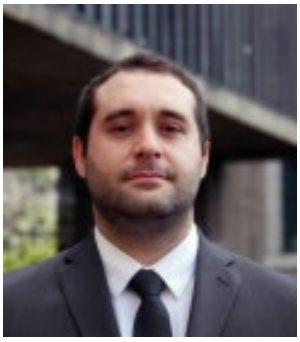
WICKHAM, Hadley. Tidy data. *Journal of Statistical Software.* v59i10, 2014, p, 1-23.

WICKHAM, Hadley. *Advanced R.* Chapman and Hall/CRC, 2014.

WICKHAM, Hadley. *ggplot2: elegant graphics for data analysis*. Springer, 2016.

PROFESSOR-AUTOR

Umberto Mignozzetti



Formação acadêmica

- Graduação em Ciências Sociais pela Universidade de São Paulo.
- Doutor em Ciência Política pela Universidade de São Paulo.
- Doutorando em Ciência Política na New York University.

Experiências profissionais

- Professor adjunto da Escola de Relações Internacionais da FGV.

FGV RI