Математические пакеты

Введение в язык программирования R

Сучков Андрей Игоревич

Санкт-Петербургский государственный электротехнический университет «ЛЭТИ»

24 октября 2020 г.

Язык программирования R

Справка

- R – язык программирования для статистической обработки данных и работы с графикой, а также свободная программная среда вычислений с открытым исходным кодом в рамках проекта GNU
- Разработан в 1993 г. Россом Айхэком и Робертотм Джентлменом
- Испытал влияние языков S и Scheme
- Название языка произошло от первых букв имён создателей (**R**oss Ihaka and **R**obert Gentleman)

Язык программирования R

Преимущества и недостатки

Преимущества:

- R – бесплатный с открытым исходным кодом
- Огромное количество пакетов и библиотек для статистической обработки и анализа данных
- Полностью программируемая высокоуровневая графика
- Обширное сообщество разработчиков

Недостатки:

- Сложен как язык программирования
- Легко написать плохой (медленный, нечитаемый) код
- Пакеты дополнений быстро устаревают

Представление и типы данных

- Типы данных:
  - Векторный
  - Списки
  - Матрицы
  - Массивы
  - Факторы
  - Таблицы данных
- Некоторые типы векторных данных:
  - `logical`
  - `integer`
  - `numeric/double`
  - `complex`
  - `character`

Листинг 1: Однострочные комментарии

```
# Single line comment
# Another line comment
```

Листинг 2: Присваивание

```
a <- 4
b <- d <- a + 42
42 -> e  # it works but it's ridiculous
# Hot key (in RStudio): Alt+-
```

Листинг 3: Простейшие арифметические операции

```
a <- 5 - 4 * (-0.5) / (19 + .25)

b1 <- 3 ^ 2
b2 <- 4 ** (-0.5)  # equivalent

d <- 5 %% 3  # return 2

e <- 11 %/% 3  # return 3
```

Листинг 4: Логические переменные и операции

```
boolT <- TRUE    # equivalent: T
boolF <- FALSE   # equivalent: F

! x       # logical NOT
x != y    # logical NOT EQUAL
x == y    # logical EQUAL
x < y     # logical LESS
x <= y    # logical LESS OR EQUAL
x > y     # logical GREAT
x >= y    # logical GREAT OR EQUAL
x & y     # logical AND
x | y     # logical OR
```

Комплексные числа

Листинг 5: Представление комплексных чисел

```
z <- 5 - 4i

x <- -2; y <- 7
w <- x + y * 1i

is.complex(w);  # return TRUE

a <- Re(z)
b <- Im(z)
z1 = Conj(z)  # complex conjugate

as.complex(y)  # return 7+0i
```

Системные переменные

Листинг 6: Некоторые системные переменные

```
x <- pi / 6
sin(x)  # return 0.5

is.infinite(Inf)  # return TRUE

y <- c(1, 2, 3, NA, 5, NA, 7)
NA == NA    # return NA
is.na(NA)   # return TRUE

NaN == NaN    # return NA
is.nan(NaN)   # return TRUE
```

Справка в R

- Чтобы получить справку о функции, воспользуйтесь функцией `help`
- Чтобы найти функцию по некоторому слову, воспользуйтесь функцией `help.search`

Листинг 7: Использование функций помощи

```
help("sin")  # get help text for sine
?sin         # equivalent

help.search("fourier")  # search for the string "fourier"
                        # in the documentation
??fourier               # equivalent
```

Создание векторов

- Для создания вектора с шагом ± 1 используется оператор двоеточие.
- Для создания вектора с произвольными значениями используется функция `c()`.
- **Важно:** нумерация элементов вектора начинается с **1**!
- Обращение к элементу вектора осуществляется в **квадратных** скобках.

Создание векторов

Примеры

Листинг 8: Создание векторов

```r
x1 <- 1:5    # return c(1, 2, 3, 4, 5)
x2 <- 2:-1   # return c(2, 1, 0, -1)

x <- c(2, -4.5, pi, -0.3, 42)

z1 <- seq(from = -3, to = 5, by = 0.5)
z2 <- seq(from = -3, to = 5, length.out = 10)

x[1]            # return 2
x[length(y)]    # return 42
x[-3]           # return c(2, -4.5, -0.3, 42)
x[x > 0]        # return c(2, pi, 42)
x[x %in% x1]    # return 2
```

Программирование

Программирование осуществляется не только в консоли, но и в скрипт-файлах (файлы с расширением .R)

Листинг 9: Файл example1.R

```
x <- 1:5
y <- c(-4, 8, 12, 3, 9)

print(x); print(y)

z <- sqrt(x)
w <- y * z

print(z); print(w)
```

Функции, определённые пользователем

Всякая функция, определяемая пользователем, имеет вид:

```
func.name <- function(x1, x2, ..., xk) {
  ## function body
  return(retval)
}
```

Важно!

Оболочка `return()` возвращает только ОДИН объект!

Листинг 10: Функция `square.R`

```
square <- function (x) {
  squared <- x * x
  return (squared)
}
```

Условный оператор и оператор альтернативного выбора

Листинг 11: Условный оператор

```
if (condition) {
  ## then-body
} else if (condition) {
  ## elseif-body
} else {
  ## else-body
}
```

Листинг 12: Оператор альтернативного выбора

```
switch (expr,
        label1 = val1,
        label2 = val2,
        ...
        label0)
```

Циклы

Листинг 13: Цикл с предусловием

```
while (condition) {
  ## loop-body
}
```

Листинг 14: Бесконечный цикл (цикл с постусловием)

```
repeat {
  ## loop-body

  if (condition) break
}
```

Листинг 15: Цикл с известным числом повторений

```
for (value in vector) {
  ## loop-body
}
```

- Матрица (matrix) – это двумерный массив данных, в котором каждый элемент имеет одинаковый тип (числовой, текстовый или логический)
- Матрицы создают при помощи функции `matrix`
- Общий формат таков:

```
mymatrix <- matrix(vector, nrow, ncol, byrow, dimnames)
```

- Обращение к элементу матрицы осуществляется в **квадратных** скобках, где через запятую указываются номер строки и столбца

Матрицы

Примеры

```
A <- matrix(1:10, nrow = 2)

A[2,]  # return c(2, 4, 6, 8, 10)

A[,2]  # return c(3, 4)

A[1,4]  # return 7

A[1,c(4, 5)]  # return c(7, 9)

t(A)  # transpose matrix

A %*% A  # matrix multiplication
```

- Массивы данных (array) сходны с матрицами, но могут иметь больше двух измерений.
- Матрицы создают при помощи функции `array`
- Общий формат таков:

```
myarray <- array(vector, dimensions, dimnames)
```

- Обращение к элементу матрицы осуществляется в **квадратных** скобках, где через запятую указываются номера размерности

Листинг 16: Пример работы с массивами

```
dim1 <- c("A1", "A2")
dim2 <- c("B1", "B2", "B3")
dim3 <- c("C1", "C2", "C3", "C4")

z <- array(1:24, c(2, 3, 4),
                 dimnames = list(dim1, dim2, dim3))

z[1,2,3]   # return 15
```