Министерство образования Республики Беларусь
Учреждение образования «Витебский государственный
университет имени П.М. Машерова»
Кафедра информационных технологий и управления бизнесом

ЯЗЫК ПРОГРАММИРОВАНИЯ R

Методические рекомендации

Витебск
ВГУ имени П.М. Машерова
2023

УДК 004.432(075.8)
ББК 32.973.22я73
Я41

Печатается по решению научно-методического совета учреждения образования «Витебский государственный университет имени П.М. Машерова». Протокол № 1 от 30.10.2023.

Составители: доцент кафедры информационных технологий и управления бизнесом ВГУ имени П.М. Машерова, кандидат биологических наук, доцент **А.А. Чиркина;** старший преподаватель кафедры информационных технологий и управления бизнесом ВГУ имени П.М. Машерова **Н.В. Булгакова**

Р е ц е н з е н т ы :
доцент кафедры прикладного и системного программирования
ВГУ имени П.М. Машерова,
кандидат физико-математических наук, доцент *С.А. Ермоченко*;
профессор кафедры информационных систем и технологий
УО «ВГТУ», доктор физико-математических наук,
профессор *А.А. Корниенко*

Язык программирования R : методические рекомендации / сост.: А.А. Чиркина, Н.В. Булгакова. – Витебск : ВГУ имени П.М. Машерова, 2023. – 32 с.

Я41

Методические рекомендации разработаны для изучения дисциплины «Язык программирования R» и ориентированы на поддержку лабораторных занятий. Данное учебное издание содержит краткие теоретические сведения и практические задания для студентов, изучающих язык программирования R и методы анализа данных. Предназначено для всех специальностей факультета математики и информационных технологий.

УДК 004.432(075.8)
ББК 32.973.22я73

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

R – свободная программная среда вычислений с открытым исходным кодом. Язык программирования R появился в 1993 году как альтернатива языку S, который широко использовался в статистике и анализе данных. Два сотрудника Оклендского университета из Новой Зеландии Росс Айхэка (Ross Ihaka) и Роберт Джентлмен (Robert Gentleman) решили создать свой собственный язык, основанный на идеях S и предназначенный для статистического анализа и графической визуализации данных, но с открытым исходным кодом.

Язык программирования R предоставляет множество преимуществ для анализа данных, статистики и визуализации, в активе этого языка – практически любые статистические методы, как классические, так и ультрасовременные. Вот некоторые из его основных преимуществ:

- мощные статистические и математические возможности: R обладает богатыми функциональными возможностями для проведения статистического анализа данных, регрессионного анализа, проверки гипотез, анализа временных рядов и многих других методов. Это делает его популярным среди статистиков и исследователей;
- большое сообщество и обширные библиотеки: R имеет активное сообщество пользователей, для него разработано множество библиотек и пакетов, позволяющих легко выполнять различные задачи анализа данных;
- графические возможности: R предоставляет мощные инструменты для создания качественных графиков и визуализации данных. С помощью пакетов ggplot2, plotly, shiny можно строить интерактивные и динамические графики и дашборды;
- поддержка структурированных данных: R поддерживает создание и использование сложных структур данных, таких как списки, матрицы, массивы, фреймы данных и т.д., что позволяет работать с реальными наборами данных, которые содержат большие объемы информации;
- бесплатный и открытый исходный код: R является бесплатным программным обеспечением с открытым исходным кодом, что позволяет бесплатно скачать, использовать и распространять его, а также вносить свой вклад в его развитие;
- интеграция с другими языками и инструментами: R обладает возможностью интеграции с другими языками программирования, такими как C, Java, Python и SQL, что позволяет использовать разнообразные инструменты и библиотеки в рамках одного анализа данных;
- активное развитие: R постоянно развивается, и в него вносятся улучшения и добавляются новые функции благодаря активному сообществу разработчиков, которые регулярно обновляют язык и пакеты, а также предоставляют хорошую документацию и техническую поддержку;

- R является интерпретируемым языком, что позволяет быстро и легко тестировать и отлаживать код, а также выполнять интерактивный анализ данных в командной строке или в среде разработки RStudio;
- широкое применение: R используется в различных областях, включая биоинформатику, финансовый анализ, социальные исследования и многое другое.

Язык R обладает множеством преимуществ для анализа данных и статистики, но также имеет и некоторые ограничения. Вот некоторые из наиболее существенных:

- недостаточная интеграция с некоторыми инструментами: в сравнении с другими языками, такими, как Python, интеграция R с некоторыми инструментами и базами данных может быть менее удобной;
- малое сообщество в некоторых областях: в некоторых отраслях и областях, таких как машинное обучение или анализ текстовых данных, Python имеет более широкое и активное сообщество с более развитой экосистемой библиотек и инструментов;
- при работе с большими объемами данных R может столкнуться с ограничениями памяти и производительности. Кроме того, R не всегда эффективно управляет памятью (создание временных объектов, отсутствие автоматической сборки мусора), что может привести к проблемам при анализе больших данных и необходимости применять стратегии оптимизации при обработке данных;
- синтаксис R может показаться непривычным и менее интуитивным для некоторых пользователей;
- узкая специализация: в отличие от языка Python, который широко используется в различных областях, R имеет более узкую специализацию в статистике и анализе данных.

Несмотря на эти ограничения, R остается мощным инструментом для анализа данных и статистики, особенно в академической среде и областях, где статистика играет ключевую роль. Разработчики R постоянно совершенствуют функциональность языка. Появляются новые пакеты и библиотеки, в которых реализованы современные методы анализа данных и статистического моделирования.

Развитие языка R активно продолжается в ответ на растущий спрос на аналитику данных и машинное обучение. R остается популярным в таких областях, как биоинформатика, геномика, экология и социальные науки, где статистический анализ данных играет важную роль.

В целом R остается важным инструментом для анализа данных и статистики, удовлетворяя растущий спрос на аналитические возможности и инструменты для работы с данными.

Тема 1. Установка, настройка среды и начало работы

Для выполнения практических заданий по программированию на языке R и лабораторных работ необходимо установить:

- **Язык R**: https://cran.rstudio.com/bin/windows/base/;
- **Графический интерфейс пользователя RStudio**: https://www.rstudio.com/products/rstudio/download/ – среда разработки для языка R.

RStudio включает в себя следующие инструменты: редактор кода, который подсвечивает синтаксис R, автоматически дополняет код, проверяет ошибки и позволяет запускать отдельные фрагменты кода или целые скрипты; консоль, в которой можно вводить команды R и получать результаты их выполнения; панель среды, которая показывает все переменные, объекты и функции, созданные в текущей сессии R, и позволяет просматривать их свойства и значения; панель пакетов, которая позволяет устанавливать, обновлять и удалять дополнительные пакеты R, а также ряд других полезных инструментов.

Далее необходимо настроить среду. Для устранения проблемы несоответствия кодировок шрифтов нужно перейти в меню, в раздел *File* и выбрать там *Reopen with Encoding* (предварительно сохранив файл). Из предложенного списка выбирать UTF-8. Для удобства можно также установить эту кодировку по умолчанию, поставив в строке *Save as default encoding for source files* галочку.

Для проверки установленной по умолчанию рабочей директории можно задать R команду в виде функции getwd() – сокращение от *get working directory*. Эту команду можно написать в окне скрипта (*File/New File/R Script*) и далее нажать сочетание клавиш Ctrl + Enter. Сочетание Ctrl + Enter запускает ту часть скрипта, на которой стоит курсор. Можно набрать эту команду прямо в консоли и нажать Enter. В обоих случаях в консоли должна появиться строка с длинным путем к существующей директории. Если необходимо изменить путь к вашей рабочей директории, то можно это сделать двумя способами:

1) перейти в меню в пункт *Session,* далее пункт Set *working Directory*, и дальше выбирать *Choose Directory*;
2) с помощью команды, которая установит рабочую директорию – функции setwd() (*set working directory*), в этом случае в качестве аргумента этой функции указывается полный путь к рабочей директории, начиная с корневого каталога, например: setwd('I:/!ЗАНЯТИЯ/Рабочие файлы').

Справку среде R можно вызвать несколькими способами:

1) можно в консоли или в окне скрипта написать знак вопроса, после чего нужно написать имя той функции, которую надо изучить. При нажатии Ctrl+Enter, появляется справка по данной функции.

2) можно использовать функцию help, которая в качестве аргумента принимает имя функции;
3) написать в окне скрипта или в консоли имя той функции, которая интересует, затем поставить текстовый курсор куда-нибудь в середину этого названия, и нажать F1. После этого специализированном окне появится справка по этой функции.

Предупреждение, warnings, это сообщение, которое обычно возникают в консоли из-за некорректного использования той или иной функции. Нередко возникают ситуации, когда при нажатии Ctrl+Enter вместо результата в консоли появляется значок плюс. Это свидетельствует о том, что код не дописан, и система ожидает от пользователя корректного ввода команды.

В языке R все функции сгруппированы в пакеты. Например, пакет под названием ggplot2 – это пакет, в котором представлены функции для построения графиков. Установка пакетов происходит обычно из сетевых ресурсов, большинство пакетов находится на сайте CRAN. Для установки пакетов используется функция *Install packages*. После установки пакетов их надо активировать с помощью функции library, в качестве аргумента для этой функции используется имя того пакета, который необходимо активировать. Пакет надо активировать лишь один раз за текущую сессию. При повторном входе в систему пакеты вновь нужно активировать. Список уже установленных пакетов можно просмотреть в специальном окне Packages, галочками помечены будут те пакеты, которые активированы в данной сессии. Поставив галочку, можно активировать пакет, не записывая команды library.

Тема 2. Основные типы данных и операции в R

Операторами присваивания в R выступают либо **=**, либо **<-** присваивание справа, либо **->** – присваивание слева. Следует иметь в виду, что в R отсутствуют скалярные структуры. Примеры простейших вычислений в консольном режиме (единица в квадратных скобках в результате – номер элемента вектора):

```
> a <- 3
> a
[1] 3
> 5 -> b;b
[1] 5
```

```
> c = 7;c
[1] 7
> c<-d<-10;c
[1] 10
> d
[1] 10
```

Значение также может быть результатом арифметического выражения:

```
> n <- sqrt(3)+4/7-1.5*13^2
> n
[1] -251.1965
```

Строки печатаются в кавычках: двойных или одинарных:

```
> "Hello world!"                > 'Hello world!'
[1] "Hello world!"              [1] 'Hello world!'
```

Логические выражения возвращают TRUE или FALSE:

```
> 3 < 4
[1] TRUE
```

Чтобы сравнить два выражения, используется двойной знак равенства:

```
> 2 + 2 == 5
[1] FALSE
```

Вектор можно задать с помощью функции c():

```
> y <- c(1, 2, 3, NA, 5)
> y
[1] 1, 2, 3, NA, 5
```

NA – это пропущенное наблюдение (от англ. Not Available). Его не следует путать с NaN (Not a Number – «не число», неопределенность).

Пример: найти среднее значение элементов вектора y:

```
> mean(y)
[1] NA
```

Необязательным аргументом функции mean является na.rm, по умолчанию равный FALSE. Как видно в этом примере, функция возвращает NA, так как она учитывает отсутствующие значения по умолчанию. Однако, если указать na.rm = TRUE в вызове функции, она игнорирует отсутствующие значения и возвращает среднее значение без учета NA.

```
> mean y, na.rm = TRUE)
[1] 2.75
```

При работе с функциями можно использовать как полные имена аргументов, так и сокращать их до первой буквы, если имя аргумента уникально среди всех аргументов этой функции. Также можно указывать аргументы по порядку, через запятую, тогда имена можно не использовать.

Пример: команда **round(x, digits = 0)** округляет значения до указанного количества знаков после запятой. Она имеет два аргумента: число, которое нужно округлить, и значение digits, сообщающее, до какого знака округлять (по умолчанию равное нулю). Команду можно написать разными способами:

```
> round(3.751, digits=0)
[1] 4
> round(3.751, d=1)
[1] 3.8
> round(d=0, 3.751)
[1] 4
> round(3.751, 2)
[1] 3,75
> round(3.751)
[1] 4
```

Посмотреть аргументы функции и их значения по умолчанию можно с помощью команды args():

```
> args(round)
function (x, digits = 0)
```

Последовательность чисел можно задать двумя способами: **start:end**, либо функцией **seq()**:

```
> 5:9
[1] 5 6 7 8 9
> seq(5,9)
[1] 5 6 7 8 9
> seq(10,50, by=10)
[1] 10 20 30 40 50
```

Обращаться к элементам вектора можно, используя квадратные скобки:

```
> sentence <- c('one', 'two', 'three')
> sentence[3]
[1] "three"
> sentence[c(1,3)]
[1] "one" "three"
```

Элементам вектора можно задать имена:

```
> ranks <- 1:3
[1] 1 2 3
> names(ranks) <- c("first", "second", "third")
> ranks
first second  third
    1      2      3
```

Можно выбрать сразу несколько интересующих элементов, указав в квадратных скобках вектор, содержащий индексы необходимых элементов:

```
> x = 2 * c(1, 2, 3, 4, 5)
> x[c(1, 3, 5)]
[1] 2 6 10
```

Вместо указания конкретных индексов, в квадратных скобках можно указать условие. Соответственно, будут отобраны только те элементы, которые удовлетворяют этому условию:

```
> x[x > 5]
[1] 6 8 10
```

Как и в случае с одиночным индексом, выражению с множественным индексом можно присваивать значения — в таком случае значения присваиваются только указанным элементам:

```
> x[c(2, 4)] = 0
> x
[1] 2 0 6 0 10
> x[x > 0] = 0
> x
[1] 0 0 0 0 0
```

Типы данных

Язык **R** работает с объектами, которые имеют два встроенных атрибута: тип данных и длина.

Тип данных – вид элемента; имеются четыре типа данных: num(числовой), char(символьный), complex(комплексный) и logical(логический).

Длина – общее количество элементов объекта.

Vector (вектор) – обычная переменная.

Factor (фактор) – категорийная переменная.

Array (массив) – многомерные массивы, чья размерность больше двух.

Matrix (матрица) – частный случай массива с размерностью, равной 2. У массива и матрицы все элементы одного и того же типа.

Data.frame – таблица, состоящая из нескольких векторов одинаковой длины, но, возможно, различных типов. Это наиболее общая структура, используемая при работе с R.

Ts – набор данных временного ряда, содержащий дополнительные атрибуты, такие как: частота и дата.

List – коллекция объектов, доступ к которым можно осуществить по номеру или имени.

В data.frame, Ts и List возможно использование нескольких типов данных. Также следует иметь в виду, что R различает в названиях объектов заглавные и строчные буквы.

Операции с векторами

Создать вектор можно следующим образом:

```
> c (1:3, seq (100, 150, 10))
[1]   1   2   3 100 110 120 130 140 150
```

С векторами можно выполнять различные арифметические действия:

```
> x <- c (1,2,3,4)
> y <- c (10,11,12,13)
> z <- x + y
> z
[1] 11.0 13.0 15.0 17.0
```

Если длина векторов различна, самый короткий вектор будет использоваться в вычислениях несколько раз, при этом длина длинного вектора должна быть кратна длине короткого вектора. Например:

```
> x <-c (1,2,3,4)
> y <-c (1,2)
> z <-x+y
> z
[1] 2 4 4 6
```

Умножить вектор на число:

```
> x <- c (1,2,3,4)
> a<- 10
```

```
> z <- a*x
> z
[1] 10 20 30 40
```

Допустимые арифметические операторы для создания выражений:

+, -, *, /, и ^ (степень), %% (x %% y – остаток от деления), %/% – деление нацело (x % / % y возвращает целую часть от деления).

Примеры функций:

Функция	Описание
sum(x)	сумма элементов объекта x (вектора)
prod(x)	произведение элементов объекта x
max(x)	максимальное значение из объекта x
min(x)	минимальное значение из объекта x
which.max(x)	индекс максимального значения объекта x
which.min(x)	индекс минимального значения объекта x
range(x)	минимальное и максимальное значения объекта x
length(x)	число элементов в объекте x
mean(x)	среднее значение элементов объекта x
median(x)	медиана объекта x
var(x)	дисперсия элементов вектора
round(x,n)	округляет элементы x до n знаков после запятой
rev(x)	перестановка элементов x в обратном порядке
sort(x)	сортирует элементы x в возрастающем порядке; сортировать в убывающем порядке rev(sort(x))

Операции с матрицами

R имеет средства для вычисления матриц и управления ими. Матрица может быть создана при помощи функции **matrix()**:

```
> matrix (1:6, nr=2, nc=3)
     [, 1] [, 2] [, 3]
[1,]    1      3      5
[2,]    2      4      6
```

Функции rbind() и cbind() связывают (объединяют) матрицы относительно строк или столбцов, соответственно:

```
> m1 <-matrix (data=1, nr=2, nc=2)
> m2 <-matrix (data=2, nr=2, nc=2)
> rbind (m1, m2)
     [,1] [,2]
[1,]    1    1
[2,]    1    1
[3,]    2    2
[4,]    2    2
```

```
> cbind (m1, m2)
   [,1] [,2] [,3] [,4]
[1,] 1    1    2    2
[2,] 1    1    2    2
```

Оператор произведения двух матриц “% * %”, например:

```
> rbind (m1, m2) %*% cbind (m1, m2)
   [,1] [,2] [,3] [,4]
[1,] 2    2    4    4
[2,] 2    2    4    4
[3,] 4    4    8    8
[4,] 4    4    8    8
> cbind (m1, m2) %*% rbind (m1, m2)
     [,1] [,2]
[1,] 10 10
[2,] 10 10
```

Транспонирование матрицы производится функцией **t**. Функция **diag()** может использоваться для построения диагональной матрицы:

```
    > diag (3)
     [,1] [,2] [,3]
[1,]  1    0    0
[2,]  0    1    0
[3,]  0    0    1
> diag (rbind (m1, m2) %*% cbind (m1, m2))
[1] 2 2 8 8
> v <-c (10,20,30)
> diag (v)
     [,1] [,2] [,3]
[1,] 10  0    0
[2,] 0   20   0
[3,] 0   0    30
```

Массив может быть создан функцией **array(<вектор-данных>, <вектор-измерений>)**. Например:

```
> x = array(1:20, c(4, 5))
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5   9    13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
```

Для многомерного массива вывод на экран выполняется срезами:

```
> x = array(1:48, c(3, 8, 2))
> x
, , 1
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]   1    4    7   10   13   16   19   22
```

```
[2,]    2    5    8   11   14   17   20   23
[3,]    3    6    9   12   15   18   21   24
, , 2
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]   25   28   31   34   37   40   43   46
[2,]   26   29   32   35   38   41   44   47
[3,]   27   30   33   36   39   42   45   48
```

Списки

Списки – важный тип представления данных. В отличие от вектора или матрицы, которые могут содержать данные только одного типа, список может состоять из элементов различных типов, в том числе и из других списков. Пример:

```
> l <- list(1:5, "Name", TRUE, list("one", 1:3))
> l
[[1]]
[1] 1 2 3 4 5
[[2]]
[1] "Name"
[[3]]
[1] TRUE
[[4]]
[[4]][[1]]
[1] "one"
[[4]][[2]]
[1] 1 2 3
```

Отображение и удаление объектов в памяти

Функция **ls()** показывает объекты, находящиеся памяти, отображая только их названия.

```
> name <- "BOOK"; a <- 100; b <- 50; c <- 0.5
> ls()
[1] "a" "b" "c" "name"
```

Чтобы показать характеристики объектов, можно использовать функцию **ls.str():**

```
> ls.str()
a :  num 100
b :  num 50
c :  num 0.5
name :  chr "BOOK"
```

Для удаления объекта из памяти, используется функция **rm()**: **rm(a)** удалит объект a, **rm(a,b)** – объекты a и b , **rm(list=ls())** – удалит все объекты.

Тема 3. Работа с данными в R

Ввод данных

Возможности системы R по вводу и редактированию данных ограничены ее создателями, которые предполагали, что для этого будут использоваться другие средства (например, блокнот или MS Excel). Поэтому подлежащие анализу объемные таблицы данных обычно подготавливаются при помощи сторонних приложений, и только потом загружаются в рабочую среду R из внешних файлов. Предпочтение при этом отдается текстовым файлам.

Правила подготовки загружаемых файлов:

- в импортируемой таблице с данными не должно быть пустых ячеек. Если некоторые значения по тем или иным причинам отсутствуют, вместо них следует ввести `NA`;
- импортируемую таблицу с данными рекомендуется преобразовать в простой текстовый файл с одним из допустимых расширений. На практике обычно используются файлы с расширением `.txt`, в которых значения переменных разделены знаками табуляции, а также файлы с расширением `.csv`, в которых значения переменных разделены запятыми;
- в качестве первой строки в импортируемой таблице рекомендуется ввести заголовки столбцов-переменных. Такая строка – удобный, но не обязательный элемент загружаемого файла. Если она отсутствует, то об этом необходимо сообщить в описании команды, которая будет управлять загрузкой файла. Все последующие строки файла в качестве первого элемента содержат заголовки строк (если таковые предусмотрены), после которых следуют значения каждой из имеющихся в таблице переменных. В именах столбцов таблицы не допускается наличие пробелов. Кроме того, имена столбцов, также, как и имена строк, не должны начинаться с точки или чисел. Во избежание связанных с кодировкой проблем все текстовые величины в импортируемых файлах рекомендуется создавать с использованием букв латинского алфавита;
- подлежащий импортированию файл рекомендуется поместить в рабочую папку программы. Чтобы выяснить путь к рабочей папке R на своем компьютере можно использовать команду **`getwd()`**; изменить рабочую директорию можно при помощи команды **`setwd()`**. При этом пользователям операционных систем Windows необходимо помнить, что для указания полных путей к файлам в программе R используется не обратный одинарный слэш (\), а прямой одинарный (/) либо двойной обратный слеш (\\).

R может читать данные, сохраненные в текстовом (ASCII) файле. Основной функцией для импортирования данных в рабочую среду R является **`read.table()`**.

Например, команда:

> gr <- read.table("grades2.txt", sep=";", head=TRUE)

загружает файл ***grades2.txt*** из рабочей папки. Для просмотра полученной таблицы и редактирования данных можно воспользоваться встроенной в R электронной таблицей, для этого надо набрать команду:

> data.entry(gr).

Функция **read.table()** позволяет настроить процесс загрузки внешних файлов, в связи с чем она имеет большое количество управляющих аргументов. У функции **read.table()** есть несколько параметров:

read.table(файл, header=FALSE, sep = "", quote = "\"'", dec = ".", row.names =, col.names =, as.is=FALSE, na.strings = "NA", skip=0, fill = ! blank.lines.skip, check.names=TRUE, strip.white=FALSE, blank.lines.skip = TRUE)

Параметр	Описание параметра
файл	Название файла пишется в кавычках, можно использоваться путь (символ \ не допустим, и должен быть заменен на /). Путь приводят либо в абсолютном виде (например, `file="C:/Temp/MyData.dat"`), либо указывают только имя импортируемого файла (например, `file = "MyData.txt"`), но при условии, что последний хранится в рабочей папке программы. В качестве имени можно также указывать полную URL-ссылку на файл, который предполагается загрузить из Internet, например: `file="http://somesite.net/YourData.csv"`
header	Служит для сообщения программе о наличии в загружаемом файле строки с заголовками столбцов. По умолчанию принимает значение `FALSE`. Если строка с заголовками столбцов имеется, этому аргументу следует присвоить значение `TRUE`
sep	Служит для указания используемого в файле разделителя значений переменных (separator – разделитель). По умолчанию предполагается, что значения переменных разделены "пустым пространством", например, в виде пробела или знака табуляции (sep = ""). В файлах формата csv значения переменных разделены запятыми, и поэтому для них sep = ","
quote	символы, используемые для обозначения переменных символьного типа
dec	символ, используемый для отделения дробной части числа. По умолчанию `sep = "."`

row.names	вектор с названиями строк, которые могут быть векторами символьного типа, или числового (или название) переменной файла (по умолчанию: 1, 2, 3...). Важно помнить, что все имена строк должны быть уникальными, т.е. одинаковые имена для двух или более строк не допускаются
col.names	вектор с названиями столбцов (V1, V2, V3...)
as.is	преобразование символьных значения в факторы (если ЛОЖЬ) или сохраняет их как символы (ИСТИНА)
na.strings	показывает какое значение строки считать отсутствием данным (по умолчанию - NA). Пустое значение в числовом столбце тоже считается отсутствием данных.
skip	число строк, которые будут пропущены перед чтением данных. Пример: `skip = 5`
fill = TRUE	если в файле все поля заполнены
check.names	если ИСТИНА, то проверяется допустимость имен переменных для **R**
strip.white	(условное выражение к sep), если ИСТИНА, то удаляются дополнительные пробелы до и после символьных переменных
blank.lines. skip	Если имеет значение TRUE то пустые строки в файле игнорируются, по умолчанию параметр имеет значение TRUE. Можно изменить значение на FALSE, которое используется совместно с параметром fill = TRUE

Функция **read.table()** имеет два варианта: **read.csv()** и **read.csv2()**. Эти форматы используются для ввода данных из файлов, имеющих определенный формат. Команды **read.csv** и **read.csv2** имеют следующие значения по умолчанию:

```
> read.csv (файл, header = TRUE, sep =,"", quote="\"",
dec = "."...)
> read.csv2 (файл, header = TRUE, sep = ";",
quote = "\"", dec = ","...)
```

Для интерактивного выбора загружаемого файла, который хранится вне рабочей папки R, можно применить вспомогательную функцию **file.choose()** (*выбрать файл*). Выполнение этой команды приводит к открытию обычного диалогового окна операционной системы Windows, в котором пользователь выбирает папку с необходимым файлом. Очень удобно совмещать **file.choose()** с командами **read.table()** или **read.csv()**, например:

```
> ch <- read.table(file = file.choose(), header
= TRUE, sep = ",")
```

В R используются еще две функции для загрузки данных.

Функция **scan()** более гибкая, чем **read.table()** и имеет больше параметров. Главное отличие в том, что, при вводе можно указать тип переменных.

Функция **read.fwf()** используется, для чтения данных имеющих определенный фиксированный формат.

Сохранение данных

Данные удобно сохранять в виде текстовых таблиц, которые потом можно будет открывать другими приложениями. Для этого служат функции **write()**, **write.table()**.

Функция ***write(x, file="data.txt")*** сохраняет объект *x* в файле *data.txt*. У данной функции имеются еще два параметра:

- *nc* (или *ncol*), – который определяет число столбцов в файле (по умолчанию *nc=1*, если *x* символьного типа, *nc=5* для других типов);
- *append* – (логического типа), если он имеет значение *TRUE*, то данные добавляются в уже существующий файл, а если имеет значение *FALSE* (значение по умолчанию), то старые данные удаляются а новые заносятся в уже пустой файл.

Функция **write.table()** сохраняет данные в файле формата *data.frame*. Формат команды:

write.table(x,файл,append=FALSE, quote=TRUE, sep="", eol = "\n", na = "NA", dec = ".", row. names=TRUE, col. names=TRUE)

Параметр	Описание параметра
sep	разделитель полей, используемый в файле
dec	символ, используемый для десятичной дроби
col.names	(логический) определяет используется ли идентификатор для названий столбцов или строк
row.names	
quote	логический или числовой вектор; если TRUE то переменные символьного типа записываются с " "; если числовой вектор, его элементы дают индексы столбцов, которые записаны в "". В обоих случаях, названия строк и столбцов также отображаются с " " если они написаны.
na	символ, который используется для пропущенных данных
eol	символ, который используется в конце каждой строки ("\n" – перевод каретки(курсора))

Например, для записи CSV-файла для ввода данных в Excel можно использовать запись:

```
> write.table(data, file = "example.csv", sep = ",", col.names = NA)
```

и для ввода этого файла обратно в R необходимо ввести команду:

```
> example <- read.table("example.csv", header = TRUE,
  sep=",", row.names=1)
```

Если надо записать во внешний файл результаты выполнения команд, используется команда **sink():**

```
> sink("1.txt", split=TRUE)
> 2+2
[1] 4
```

Тогда во внешний файл запишется строка «[1] 4», то есть результат выполнения команды. Параметр **split=TRUE**, задается для вывода данных на экран).

Тема 4. Циклы, условные операторы и функции в R

Как и многие языки программирования, R предоставляет конструкции, позволяющие управлять исполнением программы в зависимости от внешних условий.

Блоки выражений

Выражения можно объединять в блоки, ограниченные фигурными скобками; сами выражения разделяются точкой с запятой. Результатом выполнения такого блочного выражения будет результат последнего из выражений, составляющих блок:

```
> {
+ x = 10;
+ y = 25;
+ x + y;
+ }
[1] 35
```

Проверка условий

Проверка условий осуществляется следующим образом:

```
if (< условие >)
< выражение 1> else
< выражение 2>
```

<условие> может состоять из одного или нескольких логических выражений, содержащих операторы сравнения == (равно) и != (не равно), и разделённых логическими операторами && (логическое И), || (логическое ИЛИ) и ! (логическое НЕ).

Пример:

```
> grade <- c("male", "female")
> if (is.character(grade)) grade <- as.factor(grade)
> if (!is.factor(grade)) grade <- as.factor(grade) else
print("Grade already is a factor")
 [1] "Grade already is a factor"
```

В первом случае, если переменная `grade` – текстовый вектор, она преобразуется в фактор. Во втором случае выполняется одна из двух команд. Если переменная `grade` – не фактор (символ !), то она преобразуется в него. Если же эта переменная – уже фактор, то на экран выводится сообщение об этом (Grade already is a factor).

Конструкция **ifelse** – компактная и векторизованная версия конструкции `if-else`. Синтаксис:

ifelse(< условие >, < выражение 1>, < выражение 2>)

Первая команда выполняется, если условие истинно. Если условие ложно – выполняется вторая команда.

Примеры:

```
> score <- c(40,70)
> outcome <- ifelse (score > 50, "Сдал", "Провалился")
> outcome
[1] "Провалился" "Сдал"
```

Конструкция **switch** выбирает команды в зависимости от значения, которое принимает выражение:

switch(<переменная>, ...)

где многоточие означает команды, соответствующие возможным значениям *переменной*. Пример:

```
> color <- c("blue", "red")
> for (i in color)
print(
      switch(i,
         yellow = "желтый",
         red = "красный",
         blue = "синий",
         green = "зеленый"
      )
    )
[1] "синий"
[1] "красный"
```

Циклы

Циклические выражения с заданным множеством итераций выполняются при помощи конструкции

for (<переменная> in < выражение 1>)
< выражение 2>

Здесь результатом <выражения 1> должен быть вектор, <переменная> на каждой итерации цикла принимает значение очередного элемента этого вектора. Количество итераций равно количеству элементов в векторе. Соответственно, в <выражении 2> может использоваться <переменная>, которая будет переменной (счётчиком) цикла:

```
> sum = 0
> for ( i in 1:20 )
+   sum = sum + i;
> sum
[1] 210
```

В случае, когда нужно выполнять некоторую последовательность команд до тех пор, пока выполняется какое-либо условие, используется цикл:

while (<условие>) < выражение >

Пример вычисления факториала:

```
fact=function(x) {
f = 1
t = x
while(t>1) {
f = f*t
t = t-1 }
return(f) }
> x<- fact(6)
[1] 720
```

Прерывание цикла осуществляется командой **break**. Для прерывания текущей итерации и перехода к следующей служит команда **next**.

Объявление и вызов функций

Функция в R – это объект, принимающий аргументы и возвращающий некоторое значение. Общий вид объявления функции:

<переменная> = function(<аргументы>) < выражение >

Вызов функции будет осуществляться следующим образом:

<результат> = < переменная>(<аргументы >)

При вызове функции можно указывать аргументы в любом порядке, но тогда нужно явно определять, какой именно аргумент имеется в виду. Указывать непосредственно нужно только те аргументы, у которых нет значений по умолчанию. Именовать аргумент можно не полностью, а только первыми буквами, если аргумент определяется ими однозначно.

Если в рабочей среде R уже инициализированы переменные с именами, совпадающими с аргументами функции, аргументы их заместят.

Все присваивания значений переменным, которые осуществляются внутри функции, являются внутренними для этой функции и не сохраняются при выходе из неё. Для того, чтобы присвоить значение глобальной переменной внутри функции, используется конструкция **<<-**.

Существует несколько функций, которые помогают исправлять ошибки в написанных пользовательских функциях. Для создания сообщений об ошибке можно использовать функцию `warning()`. Функция `message()` создает диагностические сообщения, а функция `stop()` останавливает выполнение функции и выводит на экран сообщение об ошибке.

Ряд функций осуществляют векторизованные вычисления и, таким образом, позволяют избежать использования циклических конструкций. Это функции **apply() ,lapply(), sapply(), tapply()** и другие.

Анонимные функции

С **apply()**, **sapply()** и **lapply()** часто используют анонимные функции, пример одной из которых приведён ниже:

```
> (function(x,y){ z <- 5*x^2 + y^2; x*y+z })(0:5, 2)
[1]   4  11  28  55  92 139
```

В результате выполнения данной функции создается числовой вектор.

Команда **apply()**

Команда apply() используется тогда, когда нужно применить какую-нибудь функцию к объекту – строке или столбцу матрицы (таблицы данных). Полная форма записи: **apply (x, <указатель>, <функция>, ...)**

Здесь:

- x – имя матрицы или таблицы данных;
- указатель – указывает, к чему применяется функция: 1 – к строкам, 2 – к столбцам, c(1,2) – к строкам и столбцам одновременно;
- функция – имя применяемой функции. Если нужно применить простые операции вида + , - и т.д., то их необходимо задать в кавычках.

Пример: создать матрицу и найти сумму элементов по строкам, столбцам, корень квадратный по всем элементам матрицы.

```
> X=matrix(1:20, nrow=4)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
> apply(X,1,sum)
[1] 45 50 55 60
> apply(X,2,sum)
[1] 10  26  42  58 74
```

Длины полученных векторов соответствуют числу столбцов и строк соответственно.

Корень квадратный по всем элементам матрицы:

```
> apply(X,1,sqrt)
         [,1]     [,2]     [,3]     [,4]
[1,] 1.000000 1.414214 1.732051 2.000000
[2,] 2.236068 2.449490 2.645751 2.828427
[3,] 3.000000 3.162278 3.316625 3.464102
[4,] 3.605551 3.741657 3.872983 4.000000
[5,] 4.123106 4.242641 4.358899 4.472136
```

Команда **sapply()**

Функция sapply() применяется аналогично apply(), но только к векторам и спискам. Она полезна при сложных итерационных вычислениях, так как позволяет избежать создания циклов.

sapply (X,функция, ..., simplify = TRUE, USE.NAMES = TRUE)

- где X – объект, к которому применяется команда;
- функция – применяемая к объекту X функция;
- simplify – логический аргумент: нужно ли представлять выводимый результат в виде вектора или матрицы (значение TRUE – по умолчанию);
- USE.NAMES – логический аргумент: если данный аргумент принимает значение TRUE (по умолчанию) и X символьного типа, то в качестве названий для выводимых результатов используется X.

В результате применения **sapply()** создаётся список той же размерности, что и объект X.

Пример: создать список, элементами которого являются три числовых вектора. Найти границы диапазона значений для каждого вектора.

```
> x <- list(a = 1:10, b = exp(-3:3), c = c(1,3,5,7,10))
> x
$a
 [1]  1  2  3  4  5  6  7  8  9 10
$b
[1] 0.0497 0.1353 0.3678 1.0000 2.7182 7.3890 20.0855
$c
[1]  1  3  5  7 10
```

Теперь с помощью sapply() найдем нижнюю и верхнюю границы для вышеуказанных векторов.

```
> sapply(x, range)
      a        b     c
[1,]  1  0.04978707  1
[2,] 10 20.08553692 10
```

Команда **lapply()**

Функция **lapply(X,функция, ...)** является версией функции **sapply()**, а именно:

sapply(X,функция, ..., simplify = FALSE, USE.NAMES = FALSE)

lapply() является полезной при работе со списками, позволяет применять различные функции к элементам списка(числовые функции можно применять только в том случае, если все элементы списка X принадлежат классу `numeric`).

Пример. Создадим список из трёх компонент – символьного, числового и логического векторов:

```
> a=c("a","b","c","d")
> b=c(1,2,3,4,4,3,2,1)
> c=c(T,T,F)
> X=list(a,b,c)
> X
[[1]]
[1] "a" "b" "c" "d"
[[2]]
[1] 1 2 3 4 4 3 2 1
[[3]]
[1]  TRUE  TRUE FALSE
```

Найдем с помощью функции **lapply()** длины элементов списка и средние значения:

```
> lapply(X,length)
[[1]]
[1] 4
[[2]]
[1] 8
[[3]]
[1] 3
```

```
> lapply(X,mean)
[[1]]
[1] NA
[[2]]
[1] 2.5
[[3]]
[1] 0.6666667
```

Тема 5. Графики и диаграммы в R

Язык R является мощным инструментом для создания графиков и визуализации данных. Он предоставляет разнообразные функции и пакеты для создания статических и интерактивных графиков.

В R существует базовая система графики, которая позволяет строить такие простые графики, как гистограммы, диаграммы рассеяния, коробчатые диаграммы с использованием функций plot(), hist(), barplot() и других. позволяет. Пакеты ggplot2, lattice, plotly, shiny, leaflet обладают более широкими возможностями для визуализации данных в R, а также поддерживают создание интерактивных графиков.

Для знакомства с графическими возможностями R можно запустить демонстрационный пример:

```
> demo(graphics)
```

При вызове графической функции открывается окно с требуемым графиком. Чтобы узнать, какие графические окна открыты в настоящее время, можно использовать функцию **dev.list()**. Чтобы изменить активное окно, нужно задать его номер, например: **dev.set(2)**. Чтобы закрыть активное окно, нужно выполнить команду: **dev.off()**.

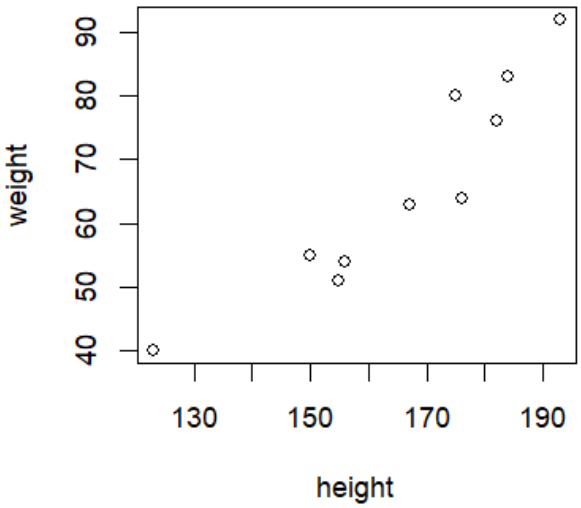
Для построения графиков используют три вида функций: функции высокого уровня (высокоуровневые); функции низкого уровня (низкоуровневые); интерактивные функции.

Графические функции (высокоуровневые)

Функции рисования высокого уровня предназначены для создания графика по данным, заданным как аргументы функции. Одна из наиболее часто используемых графических функций в R – это функция **`plot()`**.

Пример:

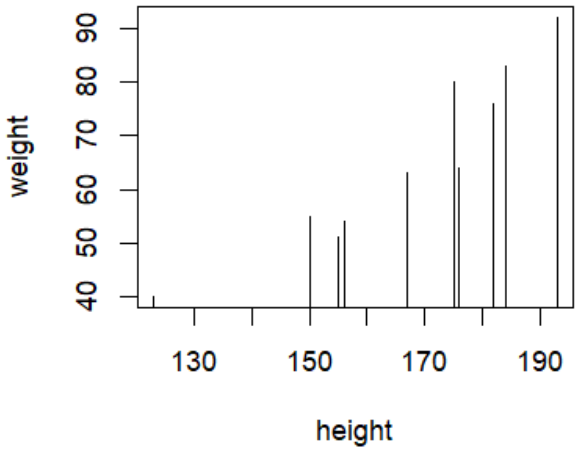
```
> height <- c(155,167,176,123,150,193,175,184,156,182)
> weight <- c(51,63,64,40,55,92,80,83,54,76)
> plot(height,weight)
```



Укажем несколько наиболее часто используемых аргументов.

Аргумент type позволяет изменять внешний вид точек на графике. Он принимает одно из следующих текстовых значений:

- "p" - точки (используется по умолчанию);
- "l" – линии;
- "b" - изображаются и точки, и линии;
- "o" - точки изображаются поверх линий;
- "h" – гистограмма;
- "s" - ступенчатая кривая;
- "n" - данные не отображаются.



Аргументы xlab и ylab служат для изменения названий осей X и Y соответственно.

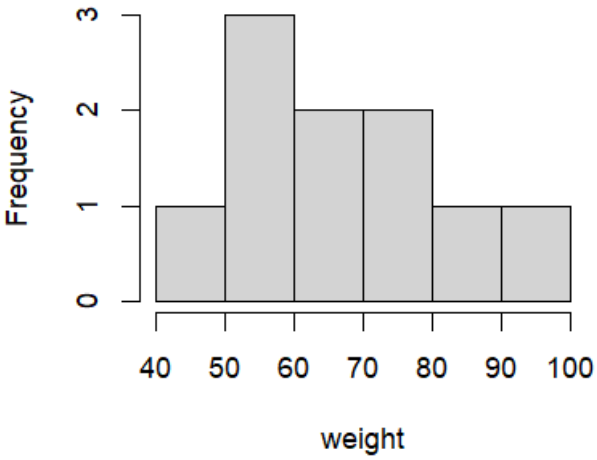
Аргументы xlim и ylim задают размах значений на каждой из осей графика. По умолчанию они оба принимают значение NULL – в этом случае размах выбирается программой автоматически.

При помощи аргумента log можно перевести одну или обе оси графика на логарифмическую шкалу

Аргумент main служит для создания названия графика. По умолчанию название размещается в верхней части рисунка.

Столбиковую диаграмму можно построить с помощью функции **barplot()**.

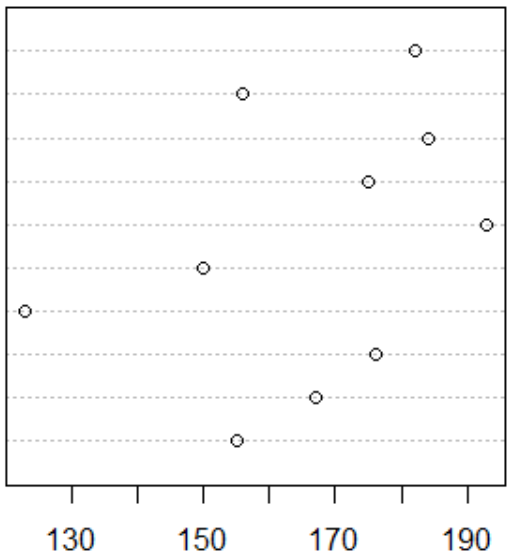
Для построения гистограммы используется функция **hist(x)**. Как правило, количество интервалов группировки выбирается автоматически, но можно их задать аргументом **nclass**. Кроме того, точки разбиения можно указать точно аргументом **breaks**. Если указан аргумент **probability = TRUE**, столбцы вместо сумм представляют относительные частоты.



Пример:

```
> hist(weight)
```

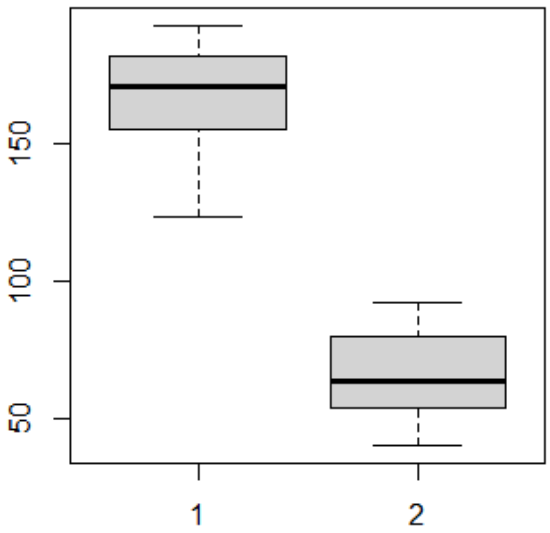
Функция **dotchart(x, ...)** создает точечную диаграмму, где по оси x отображается величина элемента данных. Это позволяет легко отобрать все записи данных, значения которых лежат в определенных диапазонах.



Пример:

```
> dotchart(height)
```

Функция **boxplot(x, ...)** создает коробчатую диаграмму («ящик с усами», «блочная диаграмма с ограничителями выбросов»), которая в удобной форме показывает медиану (или, если нужно, среднее), нижний и верхний квартили, минимальное и максимальное значение выборки и выбросы.



Несколько таких коробчатых диаграмм можно нарисовать рядом, чтобы визуально сравнивать одно распределение с другим.

Пример:

```
> boxplot(height, weight)
```

Построение функции:

```
> x = seq(-1, 1, len=100) # сто чисел в интервале [-1; 1]
> y = sin(x)  # значение функции синус в каждой из этих точек
> plot(x, y)  # длина векторов абсцисс и ординат должна совпадать
```

Основные графические функции высокого уровня:

plot(x,y)	Двумерный график
sunflowerplot(x,y)	Двумерный график (аналог plot()), но точки на графике закрашены.
piechart(x)	Круговая диаграмма
boxplot(x)	Ящик с усами
matplot(x,y)	Двумерный график первого столбца x *с* первым из y, второй из x со вторым из y, и т.д.
dotplot(x)	Если x – data.frame, производится построение Кливлендского точечного графика
pairs(x)	Если x – матрица или data.frame, рисует все возможные двумерные графики между столбцами
hist(x)	Гистограмма частот x
barplot(x)	Гистограмма значений x
qqnorm(x)	Множество значений x относительно значений, ожидаемых согласно нормальному закону распределения
persp(x,y,z)	Трехмерный график

Существует ряд параметров, которые могут быть переданы высокоуровневым графическим функциям.

Основные параметры (с их возможным значением по умолчанию):

Add	*= FALSE* *E*сли *Add=TRUE* , то новый график накладывается на текущий (если он существует)
Axes	*= TRUE* Отображение осей. *Axes=FALSE* подавляет вывод осей, что бывает полезно для того, чтобы добавить собственные оси с помощью функции axis()
Type	Определяет тип выводимого графика
Xlab **Ylab**	Подписи для осей, должны быть переменными символьного типа (любая символьная переменная, или строка в кавычках " ")
Main	Название всего графика – основной заголовок, должен быть переменной символьного типа
sub	Подзаголовок (написанный меньшим шрифтом).

Иногда высокоуровневые графические функции не дают именно такой график, какой нужен. Можно использовать низкоуровневые графические

функции, чтобы добавить дополнительную информацию (например, точки, линии или текст) к текущему рисунку.

Графические функции нижнего уровня:

Points(x,y)	Добавляет точки
Lines(x,y)	Добавляет связывающие линии к текущему графику
Text(x,y,labels,…)	Добавляет текст в рисунок в точке, заданной *labels* к координатам *(x,y)*; обычно используется*:* *plot(x,y,type="n")*; *text(x,y,names)*
Segments(x0,y0,x1,y1)	Рисует линию от точки (x0,y0) к точке (x1,y1)
Arrows(x0,y0,x1,y1, angle=30, code=2)	Аналог предыдущей с курсором в точке (x0,y0), если *code=2*, в точке (x1,y1), если *code=1* или в обоих, если *code=3*; *angle* – угол под котором будет рисоваться стрелка (указывает направление)
Abline (a, b)	Рисует линию наклона *b*, отсекающую отрезок *a*
Abline (h=y)	Рисует горизонтальную линию, может быть использовано для задания y-координаты высоты горизонтальной линии, проходящей через рисунок
Abline (v=x)	Аналогичным образом задает вертикальную линию
Abline (lm.obj)	Рисует линию регрессии (из объекта *lm.obj*). «lm.obj» может быть списком с компонентами длины 2, которые используются в качестве отсекающего отрезка и наклона (именно в таком порядке).
Rect (x1,y1,x2,y2)	Рисует прямоугольник, границы которого x1, x2, y1, y2
Poligon(x,y)	Рисует полигон, ограниченный точками с координатами x и y
Legend(x,y,legend)	Добавляет легенду к текущему графику в указанной позиции.

Title()	Добавляет заголовок и произвольно подзаголовок
Axis(side,vect)	Добавляет ось в основании (*side=1*), слева (2), наверху (3), или справа (4); *vect* (дополнительный) дает абсциссу (или ординату), куда рисуются метки
Rug(x)	Рисует данные x на *оси X* как маленькие вертикальные линии
Locator(n,type="n" ,…)	Возвращает координаты (*x, y*) после того, как пользователь нажал на графике n – время мышью; также рисует символы (*type = "p"*) или линии (*type = "l"*); по умолчанию ничего не рисует (*type = "n"*)

Последняя функция относится к построению интерактивных графиков, которые позволяют пользователям получать или добавлять информацию на график с помощью мыши. Функция **locator(n, type)** ожидает щелчка левой кнопкой мыши. Это продолжается до тех пор, пока не будет выбрано n (по умолчанию 512) точек или нажата любая другая кнопка мыши. Функция **locator()** возвращает координаты отдельных точек как список с двумя компонентами x и y.

Функцию **locator()** можно вызвать без аргументов. Это особенно полезно для интерактивного выбора позиции для графических элементов, таких, как описания и метки, когда трудно рассчитать заранее, куда надо пометить графический элемент. Например, чтобы поставить некоторый информативный текст вблизи точки-выброса, может быть полезна команда

```
> text(locator(1), "Outlier", adj=0)
```

Графические параметры

В дополнение к командам управления графиками нижнего уровня, представление графика может быть улучшено с помощью графических параметров. R поддерживает список из большого числа графических параметров, которые контролируют стиль линии, цвет, расположение рисунка и текста. Каждый графический параметр имеет имя (например, «col», который контролирует цвет) и значение (например, номер цвета).

Функция **par()** изменяет постоянные графические параметры, то есть последующие графики будут строиться относительно параметров, указанных пользователем.

Например:

> par() без параметров возвращает список всех графических параметров и их значения для текущего устройства.

> par (bg = "yellow") будет рисовать все последующие графики с желтым фоном. Исчерпывающий список графических параметров можно получить, выполнив ввод ***? par.***

Замечание: вызовы **par()** всегда влияют на значения графических параметров глобально, даже когда **par()** вызывается внутри функции. Это не всегда нужно – как правило, мы хотим установить некоторые параметры графики для некоторых рисунков, а затем восстановить исходные значения, чтобы не повлиять на пользовательскую сессию R.

Чтобы восстановить некоторые параметры, временно сохраняя результат **par()**:

```
> oldpar <- par(col=4, lty=2)
[... строим график ...]
> par(oldpar)
```

Чтобы сохранить и восстановить вообще все настраиваемые графические параметры, используем

```
> oldpar <- par(no.readonly=TRUE)
[... строим график ...]
> par(oldpar)
```

Основные графические параметры:

adj	управляет выравниванием текста (0 – выравнивание по левому краю, 0.5 – по центру, 1 – по правому краю)
bg	определяет цвет фона
bty	управляет типом поля вокруг графика, допустимые значения: *"o", "l", "7", "c", "u"* или *"]"* (поле напоминает соответствующий символ); если *bty = "n"* поле не рисуется
cex	значение регулирует размер текстов и символов относительно значения по умолчанию
col	управляет цветом символов; можно использовать *col.axis, col.lab, col.title, col.sub*
font	целое число, которое управляет стилем текста (*0 – обычный, 1 – курсив, 2 – полужирный, 3 – полужирный курсив*); можно использовать *font.axis, font.lab, font.title, font.sub*
las	целое число, которое управляет ориентацией примечаний на осях (*0 – параллельно осям, 1 – горизонтально, 2 – перпендикулярно осям, 3 – вертикально*)

lty	управляет типом линий, может быть целое число (*1 – сплошная, 2 – штриховая, 3 – пунктирная, 4 – штрих пунктирная, 5 – longdash, 6 – двойное подчеркивание*), или строка до восьми символов (между "0" и "9"), которая альтернативно определяет длину в точках или пикселях, отображая элементы и пробелы
lwd	числовой, который управляет шириной линий
mar	вектор из четырех числовых значений, которые управляют пространством между осями и границей рисунка. Формат *c (bottom,left,top,right)*, значения по умолчанию - c(5.1, 4.1, 4.1, 2.1)
mfcol	вектор *c(nr,nc)*, который делит графическое окно как матрицу из *nr* строк и *nc* столбцов, графики выводятся в столбцах
mfrow	То же самое, но графики выводятся в строках
pty	символ, который определяет тип области составления графика, *"s"* – квадрат, *"m"* – максимальное
xaxt	если *xaxt = "n"* ось *X* установлена, но не нарисована
yaxt	если *yaxt = "n"* ось *Y* установлена, но не нарисована

Таким образом, язык R является одним из самых популярных и мощных языков для визуализации данных. Он предоставляет множество функций и пакетов, которые позволяют создавать различные типы графиков и диаграмм, таких как диаграммы рассеяния, гистограммы, коробчатые диаграммы, круговые диаграммы, карты, тепловые карты и другие. С помощью языка R можно визуализировать как простые, так и сложные данные, а также добавлять интерактивность и анимацию к графикам. Язык R также позволяет настраивать внешний вид и стиль графиков, изменяя цвета, шрифты, легенды, подписи и другие параметры.

R является свободной программной средой вычислений с открытым исходным кодом в рамках проекта GNU. Язык и среда R поддерживаются и развиваются организацией R Foundation. В настоящий момент R широко используется как статистическое программное обеспечение для анализа данных и фактически стал стандартом для статистических программ.

СПИСОК ИСТОЧНИКОВ

1. Золотарюк А.В. Язык и среда программирования R: учебное пособие для студентов высших учебных заведений, обучающихся по направлениям подготовки 01.03.02 "Прикладная математика и информатика", 09.03.03 "Прикладная информатика", 38.03.01 "Экономика", 38.03.02 "Менеджмент" (квалификация (степень) "бакалавр") / А.В. Золотарюк; Финансовый университет при Правительстве Российской Федерации. – Москва: ИНФРА-М, 2020.
2. Мастицкий С.Э., Шитиков В.К. Статистический анализ и визуализация данных с помощью R. – Москва: ДМК-пресс, 2015.
3. Шипунов А.Б., Балдин Е.М., Волкова П.А., Коробейников А.И., Назарова С.А., Петров С.В., Суфиянов В.Г. Наглядная статистика. Используем R!. – Москва: ДМК Пресс, 2017.
4. Джеймс Г., Уиттон Д., Хасти Т., Тибширани Р. Введение в статистическое обучение с примерами на языке R. – Москва: ДМК-пресс, 2016.
5. Венэбльз У.Н., Смит Д.М. Введение в R: Заметки по R: среда программирования для анализа данных и графики: пер. с англ. Вер. 3.1.0. – Москва: 2014.
6. R for Data Science – [Электронный документ]. Режим доступа: https://r4ds.had.co.nz/index.html – Дата доступа: 01.11.2023.
7. W.J. Owen, The R Guide [Электронный ресурс]. Режим доступа: https://cran.r-project.org/doc/contrib/Owen-TheRGuide.pdf – Дата доступа: 01.11.2023.

Учебное издание

ЯЗЫК ПРОГРАММИРОВАНИЯ R

Методические рекомендации

Составители:
ЧИРКИНА Анна Александровна
БУЛГАКОВА Наталья Валентиновна

Технический редактор *Г.В. Разбоева*
Компьютерный дизайн *Л.В. Рудницкая*

Подписано в печать 29.12.2023. Формат 60x84 $^{1}/_{16}$. Бумага офсетная.
Усл. печ. л. 1,86. Уч.-изд. л. 1,26. Тираж 9 экз. Заказ 164.

Издатель и полиграфическое исполнение – учреждение образования
«Витебский государственный университет имени П.М. Машерова».

Свидетельство о государственной регистрации в качестве издателя,
изготовителя, распространителя печатных изданий
№ 1/255 от 31.03.2014.

Отпечатано на ризографе учреждения образования
«Витебский государственный университет имени П.М. Машерова».
210038, г. Витебск, Московский проспект, 33.