Министерство здравоохранения Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Российский национальный исследовательский
медицинский университет имени Н.И. Пирогова»
(ФГАОУ ВО РНИМУ им. Н.И. Пирогова Минздрава России
(Пироговский Университет))

Кафедра биоинформатики
Институт биомедицины (МБФ)

ВВЕДЕНИЕ В СТАТИСТИЧЕСКИЙ ЯЗЫК ПРОГРАММИРОВАНИЯ R

Учебно-методическое пособие

Москва
2025

Министерство здравоохранения Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего образования
«Российский национальный исследовательский
медицинский университет имени Н.И. Пирогова»
(ФГАОУ ВО РНИМУ им. Н.И. Пирогова Минздрава России
(Пироговский Университет))

Кафедра биоинформатики
Институт биомедицины (МБФ)

ВВЕДЕНИЕ В СТАТИСТИЧЕСКИЙ ЯЗЫК ПРОГРАММИРОВАНИЯ R

Учебно-методическое пособие

Рекомендовано Координационным советом
по образовательной политике Пироговского Университета

Москва
2025

УДК 004.42(075.8)
ББК 32.973-018я73
В24

Составители:

Иванов Сергей Михайлович — канд. биол. наук

Муравьева Елена Степановна — канд. биол. наук

Лагунин Алексей Александрович — д-р биол. наук, профессор РАН

Рецензенты:

Пятницкий Михаил Алексеевич — д-р биол. наук, ведущий научный сотрудник лаборатории анализа постгеномных данных ФГБНУ Научно-исследовательский институт биомедицинской химии имени В.Н. Ореховича

Щербо Дмитрий Сергеевич — канд. биол. наук, старший научный сотрудник лаборатории биомаркеров отдела молекулярных технологий НИИ Трансляционной медицины Пироговского Университета

В24 **Введение в статистический язык программирования R :** учебно-методическое пособие / сост. Иванов С.М., Муравьева Е.С., Лагунин А.А. — Москва : ФГАОУ ВО РНИМУ им. Н.И. Пирогова Минздрава России (Пироговский Университет), 2025. — 64 с.

ISBN 978-5-88458-770-0

В результате изучения материалов учебного пособия студенты познакомятся с языком программирования R, его синтаксисом, типами данных, переменными, операторами, базовыми функциями, импортом и сохранением информации, построением графиков и диаграмм, а также со средой программирования языка — RStudio.

Учебно-методическое пособие предназначено для студентов медицинских вузов, обучающихся по дисциплине «R, биостатистика» по специальностям 30.05.01 «Медицинская биохимия», 30.05.02 «Медицинская биофизика», 30.05.03 «Медицинская кибернетика», 06.04.01 «Биология», 12.04.04 «Биотехнические системы и технологии».

ISBN 978-5-88458-770-0

УДК 004.42(075.8)
ББК 32.973-018я73

© Составители, 2025
© Пироговский Университет, 2025

Содержание

Введение 5

Изучение возможностей RStudio 8

ТЕМА 1. Общие понятия. Работа с векторами 10
Типы данных 10
Структуры данных 10
Объявление переменных 11
Добавление комментариев 11
Операции над переменными 12
Обзор основных функций 14
Векторы и способы их создания 16
Операции над векторами 20
Практическое задание 23

ТЕМА 2. Структуры данных. Матрицы, таблицы данных, факторы и списки. Чтение и сохранение данных в файл 25
Работа с матрицами и таблицами данных 25
Понятие о факторах и списках 28
Функции для чтения и записи данных в файл 30
Импорт таблиц из файлов форматов .txt или .csv 31
Практическое задание 32

ТЕМА 3. Создание собственных функций. Операторы цикла и условия 36
Создание собственных функций 36
Операторы цикла и условия 38
Функции apply(), lapply() и sapply() 40
Практическое задание 41

ТЕМА 4. Базовая графика. Понятие о функциях высокого и низкого уровня 43
Функция высокого уровня plot() 45

Функции низкого уровня abline(), lines(), points(),
legend() и text() 47
Практическое задание 49

ТЕМА 5. Различные виды диаграмм. Гистограммы,
диаграммы размахов. Столбчатые и круговые диаграммы 51
Диаграммы размахов, гистограммы 51
Столбчатые и круговые диаграммы 56
Практическое задание 60

Рекомендуемая литература 62

ВВЕДЕНИЕ

R — это функциональный язык программирования с открытым исходным кодом. R используется для анализа данных, включая построение графиков и диаграмм. В настоящее время в свободном доступе находится более 22 тысяч пакетов, каждый из которых содержит готовые функции, в которых реализованы самые разнообразные методы и алгоритмы, предназначенные для анализа данных. Эти пакеты доступны в различных репозиториях: The Comprehensive R Archive Network (CRAN), Bioconductor и Github.

Для написания программ в R используется интерфейс командной строки, однако существуют и графические интерфейсы. Наиболее популярным является графический интерфейс среды RStudio — свободная среда разработки программного обеспечения с открытым исходным кодом для языка программирования R (https://posit.co/download/rstudio-desktop/).

R удобен с точки зрения написания кода. Чтобы писать программы на R, необязательно соблюдать чёткую структуру — можно просто вводить последовательный набор команд. Язык создавался специально для анализа данных, поэтому все конструкции синтаксиса являются достаточно ёмкими и понятными. В нём существует несколько очень хороших пакетов для визуализации данных, в том числе ggplot2, gplots и lattice.

История

В 1976 году появился первый язык программирования для статистической обработки данных и работы с графикой и стал называться S. Язык S был разработан лабораторией Bell Labs. Он был очень удобным, но распространялся только платно.

Поэтому сотрудники статистического факультета Оклендского универ-

https://posit.co/download/rstudio-desktop/

ситета Росс Айхэка и Роберт Джентлмен разработали бесплатный аналог — язык R (название языка состоит из первой буквы имён его создателей).

Создатели стремились разработать язык, который обеспечивал бы более качественный и понятный подход к анализу данных, статистике и графическим моделям.

На первых порах R использовался преимущественно в академической и научно-исследовательской среде, но сейчас он проник и в мир больших корпораций.

Сейчас R — один из наиболее бурно развивающихся статистических языков, используемых в корпоративной практике. Он используется для решения проблем статистики, финансов, страхования, аналитики и огромного количества других задач анализа данных. R, наряду с языком программирования Python, считается одним из наиболее востребованных и полезных языков программирования в биоинформатике и хемоинформатике.

Достоинства языка R

1. Удобные и понятные языковые конструкции (является плюсом для непрофессиональных программистов).
2. Базовые статистические методы реализованы в качестве стандартных функций (значительно повышает скорость разработки).
3. Есть несколько очень хороших пакетов для визуализации (можно строить и двумерную графику (диаграммы рассеяния, гистограммы, диаграммы размахов), и трёхмерные модели).
4. Для R разработано огромное количество дополнительных пакетов. Большинство пакетов собраны в специальном репозитории CRAN.
5. Для работы с R достаточно использовать консоль, однако для удобства пользователей созданы несколько сред разработки. Одной из самых удобных является среда RStudio. Её легко скачать с официального сайта (https://posit.co/download/rstudio-desktop/).
6. Язык R компилируется и работает в операционных системах Windows, MacOS, UNIX и подобных им системах (включая FreeBSD и Linux).

7. R доступен как бесплатное программное обеспечение в соответствии с условиями Стандартной общественной лицензии GNU Free Software Foundation в виде исходного кода.

Недостатки языка R

1. Низкая производительность. Однако в системе присутствуют пакеты, которые позволяют значительно повысить производительность программ.
2. Многие действия можно выполнить несколькими способами — синтаксис не всегда очевиден.
3. Далеко не все пакеты в R хорошо задокументированы.
4. Мало документации на русском языке.
5. Обновление версий языка R может приводить к прекращению работы библиотек (пакетов), написанных для предыдущих версий.

Одной из особенностей синтаксиса R является то, что индексация вектора начинается не с «0», как во многих других языках программирования, а с «1».

Материально-техническое обеспечение

Программное обеспечение:

1. ОС Windows
2. RStudio
3. Microsoft Office Word
4. Microsoft Office Excel

Оборудование:

1. Компьютерный класс
2. Доступ в Internet для загрузки пакетов R

https://posit.co/download/rstudio-desktop/

ИЗУЧЕНИЕ ВОЗМОЖНОСТЕЙ RStudio

Для начала работы в R необходимо скачать и инсталлировать программную среду (https://cran.r-project.org/bin/windows/base). В ней также можно писать команды на R, однако удобнее делать это в среде RStudio, которую можно бесплатно скачать с сайта — https://posit.co/download/rstudio-desktop/. Вначале необходимо инсталлировать программную среду R, а потом — RStudio.



https://posit.co/download/
rstudio-desktop/



https://cran.r-project.org/bin/
windows/base

Интерфейс RStudio

В RStudio можно выделить четыре основных окна (рис. 1). Нижнее левое окно представляет собой консоль, где можно по одной вводить и выполнять соответствующие команды. Однако все команды могут быть также записаны в виде скрипта, который пишется в верхнем левом окне. Скрипт может быть выполнен целиком, либо частично. В последнем случае нужно выделить мышью те строки кода, которые необходимо выполнить, и нажать кнопку «Run» (или комбинацию клавиш Ctrl + Enter).

Правое верхнее окно содержит две вкладки: «Environment» и «History». Первая вкладка содержит описание всех созданных пользователем переменных и функций, во второй вкладке представлена история команд, выполненных пользователем.

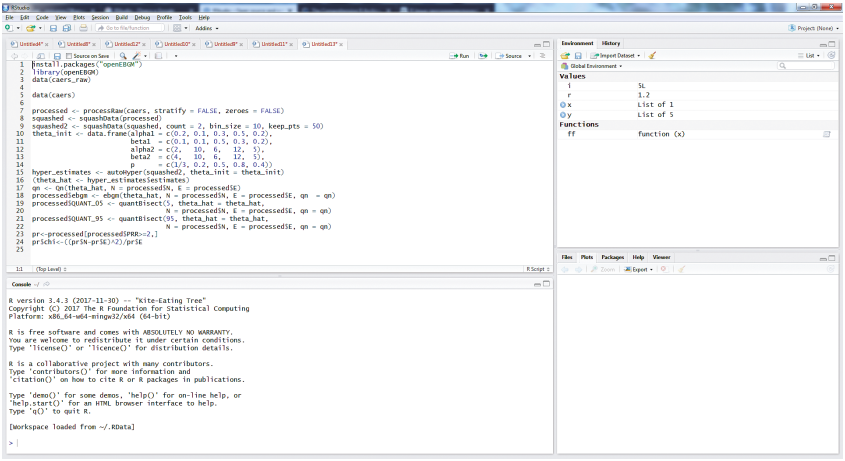


Рис. 1. Интерфейс RStudio

Правое нижнее окно содержит пять вкладок, предоставляющих различную информацию. В частности, на вкладке «Plots» будут показаны созданные пользователем графики. При закрытии RStudio можно сохранить рабочее пространство, содержащее все созданные переменные, их значения, написанные функции, чтобы в дальнейшем продолжить работу с ними (появляется соответствующее окно с предложением сохранить изменения).

ТЕМА 1.

ОБЩИЕ ПОНЯТИЯ. РАБОТА С ВЕКТОРАМИ

Типы данных

В R существует несколько типов переменных. Основными из них являются:

- числовой тип (`numeric`),
- символьный тип (`character`),
- логический тип (`logical`).

Числовой тип подразделяется на целочисленный (`integer`) и вещественный (`double`). Тип данных `integer` предназначен для хранения только целочисленных данных, тип `numeric` представляет как целые числа, так и числа с плавающей точкой. Тип данных `double` похож на `numeric`, но обладает двойной точностью, поскольку использует 64 бита для хранения значения. Тип `character` используется для хранения символьных или строковых значений в переменной. Переменные типа `logical` хранят булево значение, то есть могут принимать только два значения: `TRUE` (истина) или `FALSE` (ложь). Этот тип данных используется для представления результатов логических операций и условий.

Чтобы определить тип переменной можно использовать команду **`typeof(variable)`**.

Структуры данных

В R достаточно много структур данных. Самые важные из них:

- **Вектор (vector)** — набор (массив) элементов одного типа.
- **Матрица (matrix)** — представляют собой двумерные массивы, содержащие данные одного типа.
- **Фрейм данных (data frame)** — табличная структура для статистических операций.

▪ **Списки (lists)** — набор, в котором может содержаться комбинация типов и структур данных.

▪ **Факторы (factors)** — предназначены для представления категориальных данных.

Объявление переменных

Для того чтобы создать переменную, содержащую одно единственное значение, достаточно придумать для неё название и присвоить ей то или иное значение. Например, мы хотим создать числовую переменную `x`, хранящую значение 7, и символьную переменную `aa`, хранящую значение «`R language`». Для этого достаточно команд:

```
> x<-7
> aa<-"R language"
```

Важно запомнить, что такое окружение в R. Окружение — это место, где хранятся переменные. Это набор пар, где первый элемент — это символ (переменная), а второй — её значение. Каждый раз, когда мы объявляем переменную и вызываем её, она ищется в текущем окружении, а также рекурсивно ищется в родительских окружениях до тех пор, пока значение не будет найдено.

Для присваивания в R можно использовать и знак равенства, однако использование сочетания знаков `<-` является общепринятым. Важно отметить, что регистр в R имеет значение, то есть `x` и `X` — это разные переменные.

Добавление комментариев

Комментарии — это строчки кода, которые используются только для описания фрагмента кода и не выполняют никаких других функций. Они полезны в качестве «подсказок» для понимания кода теми, кто будет с ним работать.

Символ # используется для создания однострочных комментариев в R. В языке R нет функционала для многострочных комментариев.

Однострочный комментарий можно добавлять на одну строку вместе с основным кодом. Например:

```
print("Привет") # Этот код напечатает Привет
```

Также он может занимать и всю строку целиком. Например:

```
# Этот код напечатает Привет
print("Привет")
[1] "Привет"
```

Чтобы создать многострочный комментарий, потребуется использовать последовательные однострочные комментарии. Например:

```
# Этот код напечатает сообщение
# Привет
print("Привет")
[1] "Привет"
```

Примечание. *Выделите код, который вы собираетесь закомментировать, и нажмите Ctrl + Shift + C.*

Операции над переменными

Над числами можно выполнять обычные операции:

+ — сложение,
– — вычитание,
***** — умножение,
/ — деление,
^ — возведение в степень,
%/% — целочисленное деление,
%% — остаток от деления.

В виде команды можно записывать сложные арифметические выражения, используя круглые скобки, например:

```
> x<-7+8*(10+5-3)
```

Для того чтобы узнать полученное значение, хранящееся в переменной, можно ввести название переменной в командную строку и нажать «ввод». Значение будет выведено в консоли.

```
> x
[1] 103
```

Над *числами* и *строками* также можно выполнять операции сравнения, результатом которых будет значение «истина» или «ложь».

Можно выделить следующие операции сравнения:

- `>` — больше,
- `>=` — больше или равно,
- `<` — меньше,
- `<=` — меньше или равно,
- `==` — тождественно,
- `!=` — не тождественно.

Результат сравнения двух чисел или символов может быть записан в логическую переменную. Логические переменные могут принимать только два значения: «истина» или «ложь». Например:

```
> x<-10
> y<-11
> x<y
[1] TRUE
> L<-x<y
> L
[1] TRUE
```

Созданная переменная `L` содержит результат сравнения переменных `x` и `y`.

Обзор основных функций

Помощь

- `help(topic)`, `?topic` — справка о `topic`
- `help.search("pattern")`, `??pattern` — глобальный поиск `pattern`
- `help(package)` — справка о заданном пакете
- `help.start()` — запустить помощь в браузере
- `apropos(what)` — имена объектов, соответствующие `what`
- `args(name)` — аргументы команды `name`
- `example(topic)` — примеры использования `topic`

Текущее окружение

- `ls()` — список всех объектов
- `rm(x)` — удалить объект
- `dir()` — показать все файлы в текущей директории
- `getwd()` — получить текущую директорию
- `setwd(dir)` — поменять текущую директорию на `dir`

Общая работа с объектами

- `str(object)` — внутренняя структура объекта `object`
- `summary(object)` — общая информация об объекте `object`
- `dput(x)` — получить представление объекта в R-синтаксисе
- `head(x)` — посмотреть начальные строки объекта
- `tail(x)` — посмотреть последние строки объекта

Ввод и вывод

- `library(package)` — подключить пакет `package`
- `save(file, ...)` — сохраняет указанные объекты в двоичном XDR-формате, который не зависит от платформы
- `load()` — загружает данные, сохранённые ранее с помощью команды `save()`
- `read.table()` — считывает таблицу данных и на их основе создаёт `data.frame`
- `write.table()` — печатает объект, конвертируя в `data.frame`
- `read.csv()` — считывает csv-файл
- `read.delim()` — считывание данных, разделённых знаками табуляции

- `save.image()` — сохраняет все объекты R в файл
- `cat(…, file=, sep=)` — сохраняет аргументы, конкатенируя их через `sep`
- `sink(file)` — выводит результаты выполнения других команд в файл в режиме реального времени до момента вызова этой же команды без аргументов

В R можно вычислять значения различных функций переменных. Основные из них представлены в таблице 1.

Таблица 1. Базовые функции в R

Название функции	Описание функции
Логарифмические и экспоненциальные функции	
`exp(x)`	Нахождение экспоненты числа `x`
`log(x, base)`	Вычисление логарифма `x` по основанию `base`
`log2(x), log10(x)`	Вычисление двоичных и десятичных логарифмов `x`
Функции округления	
`round(x, dig)`	Округление числа `x`. Параметр `dig` определяет число знаков после запятой, до которого нужно провести округление
`trunck(x)`	Отбрасывает дробную часть числа
Функции модуля и квадратного корня	
`abs(x)`	Модуль числа `x`
`sqrt(x)`	Квадратный корень числа `x`
Тригонометрические функции	
`sin(x), cos(x), tan(x)`	Синус, косинус, тангенс числа `x`, представленного в радианах
`asin(x), acos(x), atan(x)`	Арксинус, арккосинус и арктангенс `x`
`sinh(x), cosh(x), tanh(x)`	Гиперболические синус, косинус и тангенс числа `x`
Другие полезные функции	
`choose(n, k)`	Число сочетаний из `n` по `k`
`factorial(x)`	Факториал числа `x`
`min(x, y), max(x, y)`	Нахождение минимума и максимума из двух чисел `x` и `y`

Для того чтобы посмотреть справку по той или иной функции, включающую описание её аргументов и результатов вычислений, введите в консоли команду **help**(«название функции») или просто ***?название функции***. Например, `?factorial`.

Векторы и способы их создания

Вектор считается одной из самых важных структур данных в R. В сущности, вектор представляет собой набор элементов, где у всех элементов должен быть одинаковый тип данных: например, только логический («истинно/ложно» — **TRUE/FALSE**), числовой или символьный.

На самом деле все переменные, созданные в примерах выше, являются векторами длиной 1. Для того чтобы создать вектор большей длины, существует несколько способов. Рассмотрим вначале наиболее простой из них, когда вектор содержит числовую последовательность.

Создание числовой последовательности. Создадим последовательность от 1 до 5 и сохраним её в вектор `x`. Это можно сделать так:

```
> x<-1:5
> x
[1] 1 2 3 4 5
```

Кроме того, создать числовую последовательность с определённым шагом можно при помощи функции **seq()**. Создадим последовательность от 1 до 5 с шагом 0,5:

```
> x<-seq(1, 5, by=0.5)
> x
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

Создание вектора с помощью функции конкатенации `c()`. Функция конкатенации «склеивает» числа и вектора в новый вектор. Например:

```
> x<-c(1, 3, 4:7, 12)
> x
[1] 1 3 4 5 6 7 12
```

Создание нового «пустого» вектора. Если необходимо создать вектор определённой длины сейчас, а заполнить его потом, то это можно сделать следующим способом:

```
> x<-numeric(5)
> x
[1] 0 0 0 0 0
```

Для создания указывается тип вектора (`numeric, character, logical`) и его длина. Чтобы узнать длину созданного ранее вектора, можно воспользоваться функцией **`length()`**:

```
> length(x)
[1] 5
```

В примере выше создан числовой вектор длиной 5, по умолчанию он заполняется нулями. Теперь можно обращаться к элементам вектора по индексам, заключённым в квадратные скобки. Например, присвоим второму элементу вектора `x` значение 7:

```
> x[2]<-7
> x
[1] 0 7 0 0 0
```

Интересно, что в R для обращения к элементам вектора можно использовать векторы индексов. В этом случае получим новый вектор, содержащий соответствующие элементы исходного. Также можно использовать отрицательные индексы, в таком случае новый вектор будет содержать все элементы, кроме тех, что соответствуют этим индексам. Проиллюстрируем это на примере. Создадим числовую последовательность от 3 до 7, сохраним в вектор, а потом пересохраним в ту же переменную,

оставив элементы с индексами 2 и 4. Сделать это можно двумя способами:

```
> x<-3:7
> x<-x[c(2, 4)]
> x
[1] 4 6
```

или

```
> y<-3:7
> y<-y[c(-1, -3, -5)]
> y
[1] 4 6
```

Каждому элементу вектора также можно присвоить имя и в дальнейшем обращаться к элементу по его имени. Присвоить имя элементам вектора можно с помощью функции **names(название вектора)**. Создадим вектор длиной 5, а потом присвоим в качестве имён буквы латинского алфавита:

```
> x<-1:5
> names(x)<-c("a", "b", "c", "d", "e")
> x
a b c d e
1 2 3 4 5
```

Имя определённого элемента вектора можно поменять, указав его индекс в квадратных скобках, после функции `names()`:

```
> names(x)[2]<-"f"
> x
a f c d e
1 2 3 4 5
```

Теперь обращаться к элементам вектора можно по их именам. Для этого нужно просто указать вместо индекса имя или вектор имён:

```
> x["f"]
f
2
> x[c("f", "e")]
f e
2 5
```

Вместо индексов можно также записывать в квадратных скобках логические выражения. Создадим вектор, представляющий собой числовую последовательность от 1 до 10, и сохраним в новый вектор те элементы, значения которых больше 5:

```
> x<-1:10
> x>5
[1] FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE
> y<-x[x>5]
> y
[1] 6 7 8 9 10
```

Результатом выражения `x>5` является логический вектор, элементы которого принимают значения «истина» или «ложь» в зависимости от того, выполняется ли это условие для соответствующего элемента вектора `x`. Логический вектор можно сохранить в новую переменную, а потом подставить её в квадратные скобки переменной `x`. Однако проще сразу записать выражение `x>5` в квадратные скобки, как это показано в примере. В результате в новый вектор `y` сохранятся те элементы `x`, которым соответствует значение «истина».

Если вы хотите найти индексы элементов, значения которых удовлетворяют условию, то можно использовать функцию **`which()`**:

```
> x<-(-5):5
> x
[1] -5 -4 -3 -2 -1 0 1 2 3 4 5
> which(x>0)
[1] 7 8 9 10 11
```

В данном примере положительными являются элементы вектора `x` с индексами с 7-го по 11-й.

Операции над векторами

Векторы, как и числа (скаляры), можно складывать, вычитать, делить и умножать. Например, если умножить вектор на число, то каждый элемент вектора будет умножен на это число. Если складывать два вектора одинаковой длины, то будут складываться элементы с одинаковыми индексами, например:

```
> x<-1:5
> y<-(-1):3
> x
[1] 1 2 3 4 5
> y
[1] -1 0 1 2 3
> x+y
[1] 0 2 4 6 8
```

Если складывать векторы разной длины, то элементы более короткого вектора дублируются необходимое число раз, чтобы его длина была равна длине более длинного вектора. В качестве примера создадим вектор `x` длиной 7 и вектор `y` длиной 3, а затем выполним сложение:

```
> x<-1:7
> y<-c(2, 4, 6)
> x
[1] 1 2 3 4 5 6 7
> y
[1] 2 4 6
> x+y
[1] 3 6 9 6 9 12 9

Warning message:
In x + y : longer object length is not a multiple of
shorter object length
```

Элементы вектора `y` будут дублированы следующим образом: 2 4 6 2 4 6 2. Результат сложения будет выведен на кон-

соль, но вместе с предупреждением о том, что длины векторов не равны.

К векторам применимы и многие функции, перечисленные в таблице 1, например, функция вычисления десятичного логарифма `log10(x)`. В этом случае логарифм будет вычислен от каждого элемента вектора `x`, и результатом вычислений также будет вектор:

```
> x<-c(1, 10, 100, 1000, 10000)
> x
[1] 1 10 100 1000 10000
> log10(x)
[1] 0 1 2 3 4
```

К векторам можно применять также функции **`min(x)`**, **`max(x)`**, **`median(x)`**, **`mean(x)`**, вычисляющие минимальное и максимальное значения, медиану и среднее элементов вектора.

Вектора определённых типов можно *конвертировать* в другие. Например, числовой вектор можно конвертировать в символьный, а логический — в числовой. В последнем случае новый вектор будет содержать значения 0 и 1. Для конвертации типов векторов используются специальные функции **`as.numeric()`**, **`as.character()`**, **`as.logical()`**. Проверить тип вектора можно с помощью функций **`is.numeric()`**, **`is.character()`** и **`is.logical()`**.

Индексирование

Векторы

- `x[n]` — `n`-й элемент
- `x[-n]` — все элементы, кроме `n`-го
- `x[1:n]` — первые `n` элементов
- `x[-(1:n)]` — все элементы, кроме первых `n`
- `x[c(1,4,2)]` — элементы с заданными индексами
- `x["name"]` — элемент с заданным именем
- `x[x > 3]` — все элементы, которые больше 3
- `x[x > 3 & x < 5]` — все элементы между 3 и 5
- `x[x %in% c("a", "and", "the")]` — все элементы из заданного множества

Списки

- `x[n]` — список, состоящий из элемента `n`
- `x[[n]]` — `n`-й элемент списка
- `x[["name"]]` — элемент списка с именем `name`
- `x$name` — элемент списка с именем `name`

Матрицы

- `x[i, j]` — элемент на пересечении `i`-й строки и `j`-го столбца
- `x[i,]` — `i`-я строка
- `x[,j]` — `j`-й столбец
- `x[,c(1,3)]` — заданное подмножество столбцов
- `x["name",]` — строка с именем `name`

Фреймы

- `x[["name"]]` — столбец с именем `name`
- `x$name` — столбец с именем `name`

Работа с переменными

- `as.array(x)`, `as.data.frame(x)`, `as.numeric(x)`, `as.logical(x)`, `as.complex(x)`, `as.character(x)` — преобразование переменной к заданному типу
- `is.na(x)`, `is.numeric(x)`, `is.null(x)`, `is.array(x)`, `is.data.frame(x)`, `is.complex(x)`, `is.character(x)` — проверка на то, что данный объект относится к указанному типу
- `length(x)` — число элементов в `x`
- `dim(x)` — размерности объекта `x`
- `dimnames(x)` — имена размерностей объекта `x`
- `names(x)` — имена элементов объекта `x`
- `nrow(x)` — число строк `x`
- `ncol(x)` — число столбцов `x`
- `class(x)` — класс объекта `x`
- `unclass(x)` — удаляет атрибут класса у объекта `x`
- `attr(x, which)` — атрибут `which` объекта `x`
- `attributes(obj)` — список атрибутов объекта `obj`

Математика

- `sin(x)`, `cos(x)`, `tan(x)`, `asin(x)`, `acos(x)`, `atan(x)`, `atan2(y, x)`, `log(x)`, `log(x, base)`, `log10(x)`, `exp(x)` — элементарные математические функции

- `min(x)`, `max(x)` — минимальный и максимальный элементы объекта `x`
- `range(x)` — вектор из минимального и максимального элемента объекта `x`
- `pmin(x, y)`, `pmax(x, y)` — возвращают вектор с минимальными (максимальными) для каждой пары `x[i]`, `y[i]`
- `sum(x)` — сумма элементов объекта `x`
- `prod(x)` — произведение элементов объекта `x`
- `diff(x)` — возвращает вектор из разниц между соседними элементами
- `mean(x)` — среднее арифметическое элементов объекта `x`
- `median(x)` — медиана (средний элемент) объекта `x`
- `weighted.mean(x, w)` — средневзвешенное объекта `x` (`w` определяет веса)
- `round(x, n)` — округляет `x` до `n` знаков после запятой
- `cumsum(x)`, `cumprod(x)`, `cummin(x)`, `cummax(x)` — кумулятивные суммы, произведения, минимумы и максимумы вектора `x` (`i`-й элемент содержит статистику по элементам `x[1:i]`)
- `union(x, y)`, `intersect(x, y)`, `setdiff(x, y)`, `setequal(x, y)`, `is.element(el, set)` — операции над множествами: объединение, пересечение, разность, сравнение, принадлежность
- `choose(n, k)` — количество сочетаний C_n^k из `n` по `k`
- `rank(x)` — ранжирует элементы объекта

Практическое задание

1. Создайте числовой вектор из 10 элементов, содержащий значения от -4 до 5.

Создайте символьный вектор длиной 5, содержащий отдельные символы, слова и короткие предложения.

Создайте числовой вектор длиной 12, содержащий повторяющуюся последовательность 101. Используйте для этого функцию `rep()`. Справку по этой функции можно вызвать при помощи команды `?rep`.

Конвертируйте полученные вектора в вектора других типов данных.

2. Создайте вектор случайных чисел при помощи функции `runif()`. Посмотрите справку по этой функции (`?runif`). Создайте вектор длиной 50, содержащий вещественные числа в диапазоне от -20 до 100.

Найдите индексы элементов вектора, значения которых больше 70.

Найдите элементы вектора, значения которых удовлетворяют следующему неравенству: 10 < значение < 70. Сохраните эти значения в новый вектор.

Поменяйте все отрицательные значения элементов исходного вектора на равные по модулю положительные. Вычислите логарифм по основанию 2 от полученного вектора и сохраните результат в новый вектор.

Добавьте названия элементов полученного вектора. Можно использовать символы латинского алфавита. Для этого возьмите имеющийся в R вектор `letters`. Например, второй элемент этого вектора:

```
> letters[2]
[1] "b"
```

Найдите элементы вектора с названиями «`e`» и «`f`». Сохраните их в новый вектор.

Посчитайте сумму, произведение и среднее элементов вектора с помощью функций `sum()`, `prod()`, `mean()`.

Вопросы

1. Какие типы данных существуют в R?
2. Какими способами можно создавать вектора в R?
3. Каким образом можно обращаться к элементам вектора?
4. Какие операции можно проводить над векторами?

ТЕМА 2.

СТРУКТУРЫ ДАННЫХ. МАТРИЦЫ, ТАБЛИЦЫ ДАННЫХ, ФАКТОРЫ И СПИСКИ. ЧТЕНИЕ И СОХРАНЕНИЕ ДАННЫХ В ФАЙЛ

Работа с матрицами и таблицами данных

Матрицы представляют собой двумерные массивы, содержащие данные одного типа. Создать матрицу из вектора `x` можно при помощи функции **`matrix(x, nrow, ncol, byrow)`**, указав число строк и столбцов, заполняя её по строкам (параметр `byrow=TRUE`) или по столбцам. Создадим матрицу из вектора длиной 9, заполнив её по строкам:

```
> mm<-matrix(1:9, nrow=3, ncol=3, byrow=T)
> mm
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

Для того чтобы создать «пустую» матрицу, содержащую нули, можно вместо вектора из девяти элементов указать скаляр 0:

```
> mm<-matrix(0, nrow=3, ncol=3, byrow=T)
> mm
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
[3,]    0    0    0
```

Присвоить какое-либо значение элементу матрицы можно, задав индексы элемента. Первый индекс представляет собой номер строки, а второй — номер столбца. Например:

```
> mm[2, 2]<-7
> mm
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    7    0
[3,]    0    0    0
```

Для обращения к элементам матрицы также можно использовать векторы индексов, векторы имён строк и столбцов, логические векторы. Имена строк и столбцов можно задать с помощью функций **`rownames()`** и **`colnames()`** соответственно.

Функции **`nrow()`**, **`ncol()`** и **`dim()`** возвращают число строк, число столбцов и размерность матрицы.

В R часто используются функции **`cbind()`** и **`rbind()`**, «склеивающие» две или более матриц в новую матрицу. Функция **`cbind(A, B)`** создаёт матрицу из матриц (векторов), приписывая справа к `A` матрицу (вектор) `B`, при этом число строк у обеих матриц должно совпадать. Функция **`rbind(A, B)`** создаёт матрицу из матриц (векторов), приписывая снизу к `A` матрицу (вектор) `B`, при этом число столбцов у обеих матриц должно совпадать.

Матрицы можно складывать, вычитать и умножать в соответствии с правилами сложения и умножения матриц. Сложение и вычитание матриц можно выполнить при помощи операторов + и – , а матричное произведение — при помощи оператора %*%. К матрицам, как и к векторам, можно применять различные функции, в том числе многие из перечисленных в таблице 1. Например, выражение **`log10(mm)`**, где `mm` — матрица, позволит получить новую матрицу той же размерности, где каждый элемент представляет собой логарифм от исходного элемента.

Существуют и *специальные функции* для проведения операций над матрицами:

- **`det(A)`** вычисляет определитель матрицы,
- **`solve(A)`** вычисляет обратную матрицу,
- **`t(A)`** транспонирует матрицу,
- **`solve(A, b)`** находит решение системы линейных уравнений `Ax=b`.

Помимо числовых матриц, могут быть также символьные и логические матрицы, но каждая матрица может содержать элементы только одного типа.

Таблица данных (фрейм данных), в отличие от матриц, может содержать столбцы разного типа, например, часть столбцов может быть числовыми, а часть — символьными. Соответственно, над таблицами данных нельзя выполнять многие операции, применимые к матрицам, например, умножение, вычисление определителя и прочее. Создать таблицу данных можно из векторов или матриц.

Создадим фрейм данных `ff` из символьного вектора `aa` и числовой матрицы `mm`.

```
> aa<-letters[1:3]
> mm<-matrix(1:9, ncol=3)
> ff<-data.frame(aa, mm)
> ff
  aa X1 X2 X3
1  a  1  4  7
2  b  2  5  8
3  c  3  6  9
```

Столбцам таблицы автоматически присваиваются имена (в примере выше это `X1`, `X2` и `X3`). Используя символ `$`, можно обратиться к определённому столбцу таблицы данных `ff`:

```
> ff$X2
[1] 4 5 6
```

Имена строк и столбцов таблицы можно изменить с помощью функций **`rownames()`** и **`colnames()`** соответственно.

Понятие о факторах и списках

Фактор — это векторный объект, который используется в R для хранения *категориальных* данных, таких как пол пациентов, тип лечения (лекарство или плацебо) и др. Эти данные также можно сохранить в виде символьного вектора. Чем отличается символьный вектор от фактора? Рассмотрим это на конкретном примере. Создадим символьный вектор `gender`, содержащий указание пола пациентов: `M` и `F`. С помощью функции **`factor()`** создадим фактор `ff` на его основе.

```
> gender<-c("M", "M", "F", "M", "F", "F")
> ff<-factor(gender)
> ff
[1] M M F M F F
Levels: F M
> ff[1]
[1] M
Levels: F M
```

Элементы фактора `ff` те же, что и у вектора `gender`, однако они уже не представляют собой символы. Кроме того, в `ff` хранятся уровни фактора: `F` и `M`. Уровень фактора показывает, какие значения может принимать категориальная переменная. В данном случае `ff` может содержать элементы, принимающие одно из двух значений: `F` и `M`.

Список представляет собой коллекцию объектов разного типа. Он может содержать векторы, матрицы, таблицы данных, факторы и другие списки, доступ к которым можно производить по индексу или имени. Создать список можно при помощи функции **`list(object1, object2, ...)`**. Рассмотрим пример, где список содержит вектор, матрицу и таблицу данных.

```
> xx<-letters[1:5]
> mm<-matrix(1:9, ncol=3)
> ff<-data.frame(aa=letters[1:10], bb=1:10, cc=seq(-9, 0, by=1))
> ll<-list(v=xx, m=mm, df=ff)
```

Запись `v=xx` при создании списков и таблиц данных означает, что соответствующий элемент списка или столбец таблицы будет создаваться на основе вектора `xx` и иметь имя `v`. Для того чтобы обратится к тому или иному объекту списка, нужно указать его индекс или имя в двойных квадратных скобках. Например, чтобы получить символьный вектор `xx`, являющийся первым элементом списка, нужно указать:

```
> ll[[1]]
[1] "a" "b" "c" "d" "e"
> ll[["v"]]
[1] "a" "b" "c" "d" "e"
```

Вектор также можно получить, указав его имя в списке после символа `$`:

```
> ll$v
[1] "a" "b" "c" "d" "e"
```

Для того чтобы получить элемент вектора, матрицы или таблицы, входящих в состав списка, нужно сначала указать индекс объекта в двойных квадратных скобках, а потом в одинарных квадратных скобках указать индекс элемента. Можно к объекту обратиться и по его имени. Например, мы хотим получить второй элемент вектора `xx`, входящего в список `ll`:

```
> ll[[1]][2]
[1] "b"
```

То же самое можно сделать следующим образом:

```
> ll$v[2]
[1] "b"
```

Списки используются в R для хранения разнородных данных и часто являются результатом выполнения какой-либо функции, возвращающей множество разных объектов (скаляры, векторы, матрицы и прочее).

Функции для чтения и записи данных в файл

Наиболее часто в ходе статистического анализа приходится иметь дело с табличными данными, поэтому в R реализованы функции сохранения табличных данных в файл и чтения их из файла. Существуют две универсальные функции **read.table()** и **write.table()**, которые позволяют читать и записывать данные.

Основные аргументы функции `read.table()` перечислены в таблице 2. Существуют также специализированные функции **read.delim()** и **read.csv()**, где значения некоторых аргументов установлены по умолчанию. Эти функции позволяют читать файлы соответствующих форматов.

Основные аргументы функции **write.table()** перечислены в таблице 3. Специализированная функция **write.csv()** записывает таблицу в csv-файл.

R предлагает для работы ряд пакетов, которые позволяют читать и записывать внешние данные. Например, Excel-файлы и таблицы SQL. Для загрузки файлов в форматах программы Excel (.xls или .xlsx) существует R-пакет «**xlsx**».

Сначала надо загрузить пакет «xlsx» в среду R:

```
> install.packages("xlsx", dep=T)
```

Затем подключить этот пакет при помощи команды:

```
> library("xlsx")
```

Далее прописываем путь и имя файла, указываем номер листа, в котором находится таблица:

```
> имя таблицы в R <- read.xlsx("путь к файлу/имя файла.xlsx",
    sheetIndex = "номер листа")
```

Примечание. *Чтобы загрузить Excel-файл в среду R, к имени файла надо обязательно добавить после точки расширение .xlsx.*

Таблица 2. Основные аргументы функции `read.table()`

Аргумент	Описание
`file`	Имя файла
`header`	При значении `TRUE` содержимое первой строки рассматривается как названия столбцов
`sep`	Разделитель между столбцами
`quote`	Вид кавычек (двойные или одинарные)
`dec`	Десятичный разделитель в числах (точка или запятая)
`row.names`	Вектор имён строк итоговой таблицы либо число — номер столбца исходной таблицы с названиями строк
`col.names`	Вектор имён столбцов итоговой таблицы либо число — номер строки исходной таблицы с названиями столбцов
`as.is`	Логический параметр, определяющий нужно ли символьные столбцы конвертировать в факторы

Таблица 3. Основные аргументы функции `write.table()`

Аргумент	Описание
`file`	Имя файла
`append`	При значении аргумента `TRUE` новые данные добавляются к исходному файлу, при `FALSE` — записываются вместо старых
`sep`	Разделитель между столбцами
`quote`	Если значение аргумента равно `TRUE`, то все символьные переменные и факторы будут записаны в файл в кавычках
`dec`	Десятичный разделитель в числах (точка или запятая)
`row.names`	Аргумент либо логический (при значении `TRUE` записываются имена строк таблицы), либо символьный, представляющий собой вектор названий строк
`col.names`	Аргумент либо логический (при значении `TRUE` записываются имена столбцов таблицы), либо символьный, представляющий собой вектор названий столбцов

Импорт таблиц из файлов форматов .txt или .csv

Большинство пользователей R предпочитают пользоваться другими вариантами импорта файлов в среду. В этих вариантах: 1) не нужно подключать в R дополнительных пакетов;

2) множество «сырых» данных сохраняется именно в форматах .txt и .csv; 3) эти форматы имеют меньший размер по сравнению с форматами .xls и .xlsx и в них нет никаких ограничений по количеству строк или столбцов.

Файлы *.txt и *.csv можно создать прямо в программе Excel. Для этого необходимо:

1. Нажать на вкладку «Файл» в левом верхнем углу.
2. Выбрать «Экспорт» и найти опцию «Изменить тип файла».
3. Щелкнуть по «Изменить тип файла», появится список различных форматов.
4. Выбрать .txt (разделитель — знак табуляции «tab») или .csv (разделитель — запятая).
5. Прописать расположение и имя файла.

Теперь необходимо импортировать полученные файлы в среду R. Для этого воспользуемся командами:

```
# для.csv:
> имя таблицы в R <- read.csv("путь к файлу/имя файла.csv",
    sep = ";")
# для.txt:
> имя таблицы в R <- read.table("путь к файлу/имя файла.txt",
    sep = "\t")
```

Заметим, что в этих случаях после имени файла также прописывается его расширение, .csv или .txt соответственно.

Практическое задание

1. Создайте 2 вектора случайных чисел длиной 25, содержащих любые вещественные числа. Преобразуйте эти векторы в квадратные матрицы (5 строк и 5 столбцов).

Найдите средние значения и суммы элементов матриц по столбцам (используйте функции `colMeans()`, `colSums()`).

Найдите определители матриц. Для этого используйте функцию `det()`.

Выполните операции сложения, вычитания и умножения матриц (для умножения матриц используйте оператор %*%).

Добавьте названия строк и столбцов к каждой матрице. Используйте символы латинского алфавита или любые слова.

Сохраните в отдельную переменную матрицу, которая является результатом пересечения любых трёх строк и двух столбцов исходной матрицы (любой из двух).

Сохраните в отдельную переменную 2-й столбец исходной матрицы.

Сохраните в отдельную переменную 4-ю строку исходной матрицы.

2. Дана система из трёх линейных уравнений:

$$4x_1 + 3x_2 - x_3 = 7,$$
$$5x_1 - 2x_2 + 4x_3 = 10,$$
$$x_1 + 2x_2 = 12.$$

Найдите определитель основной матрицы системы (матрица из коэффициентов при неизвестных). Имеет ли система единственное решение? Если да, то найдите решение с помощью **формул Крамера**: $x_i = \Delta_i/\Delta$, где Δ_i — определитель матрицы, полученной путём замены `i`-го столбца в основной матрице системы столбцом свободных членов.

Найдите решение этой же системы уравнений при помощи функции `solve()`. Сравните полученные результаты.

3. Создайте таблицу данных `mytable`, содержащую три столбца: `age`, `gender` и `disease`; и 10 строк. Сделайте так, чтобы первый столбец был числовым, второй — символьным, а третий — логическим. Заполните таблицу произвольными данными. Например:

```
  age gender disease
1  25      F    TRUE
2  35      M   FALSE
3  75      F    TRUE
```

Конвертируйте символьный столбец в фактор при помощи функции `as.factor()`.

Сохраните полученную таблицу в файл.

4. Предположим, проводилось исследование зависимости терапевтического эффекта лекарства (выраженного в процентах) от различных факторов: возраст, пол и наличие мутации в гене, кодирующем рецептор, с которым связывается лекарственное соединение. Импортируйте соответствующие данные из файла «**Response2Drug.txt»** и сохраните их в переменную. При импорте сделайте так, чтобы первый столбец был сохранён как названия строк. Откройте таблицу в RStudio, используя название переменной в окне «Environment». Ознакомьтесь с данными в таблице.

Сделайте так, чтобы названиями строк был не номер пациента «1», «2» и т. д., а «Patient 1», «Patient 2» и т. д. (Сделайте это программно, используя функцию `paste()`.)

Удалите из таблицы строки, которые не содержат указание возраста. Используйте для этого функцию `is.na()`.

Сделайте последний столбец (`Mutation`) логическим.

Посчитайте основную статистику для столбцов с возрастом (`age`) и эффектом лекарства (`response`). Используйте для этого функцию `summary()`. Каков средний возраст пациентов в исследовании?

Найдите индексы строк для трёх диапазонов возраста: <35 лет, 35–60 лет и >60 лет. Посчитайте статистику для колонки `response` для каждого диапазона возраста. Посчитайте также статистику отдельно для мужчин и женщин, а также для пациентов с мутацией (без мутации) в гене, кодирующем рецептор — мишень для лекарства. От каких факторов зависит эффект лекарства (возраст, пол, мутация в гене)?

5. Импортируйте данные из файла «**Adverse reactions.txt»** и сохраните их в переменную. При импорте укажите, что первый столбец содержит названия строк, а первая строка — это заголовки столбцов. Эта таблица содержит данные о частоте побочных эффектов при разных дозах индометацина (TIVORBEX) и в группе плацебо. Используя данные `age` и `gender` из задания 3, создайте переменную типа список, содержащую векторы `age`, `gender` и таблицу `Adverse reactions`. Теперь эта переменная содержит результаты клинических исследований по побочным эффектам и информацию об участвующих в них пациентах.

Сколько пациентов мужского пола старше 50 участвовало в исследовании? Найдите побочные эффекты, частота которых выше чем в группе плацебо. Отличаются ли эти эффекты при разных дозировках индометацина?

Вопросы

1. Какие классы данных есть в R?
2. Чем отличаются матрицы и таблицы данных?
3. Что представляет собой переменная типа фактор?
4. Что представляет собой переменная типа список?

ТЕМА 3.

СОЗДАНИЕ СОБСТВЕННЫХ ФУНКЦИЙ. ОПЕРАТОРЫ ЦИКЛА И УСЛОВИЯ

Создание собственных функций

В R, как и в других языках программирования, можно создавать собственные функции. Создать новую функцию можно следующим способом:

```
название функции <- function(аргументы) {тело функции}
```

Здесь **function** — ключевое слово, сообщающее о том, что будет создана функция, аргументы — список формальных аргументов функции, «тело функции» — заключённая в фигурные скобки команда или блок команд, зависящие от определённых ранее аргументов. Соответствующие команды в теле функции можно записывать с новой строки или через точку с запятой. Результатом выполнения функции будет результат последнего выражения, записанного в теле функции.

Например, напишем функцию, которая складывает два числа и берёт квадратный корень от суммы:

```
> ff<-function(x, y) {z<-x+y; z<-sqrt(z)}
> x<-ff(3, 6)
> x
[1] 3
```

Результатом, который выдаёт функция, может быть число, вектор, матрица, таблица данных или список. Если результатом выполнения алгоритма, реализованного в теле функции, являются несколько векторов, матриц, таблиц и прочее, то все эти

переменные нужно сохранить как элементы списка, который будет выводиться как результат вычислений. Создадим функцию, находящую сумму и разность элементов двух векторов:

```
> ff<-function(x1, x2) {
    sm<-x1+x2
    dif<-abs(x1-x2)
    ll<-list(sm=sm, dif=dif)
    return(ll)
  }

> x1<-1:5
> x1
[1] 1 2 3 4 5

> x2<-6:10
> x2
[1]  6  7  8  9 10

> mm<-ff(x1, x2)

> mm$sm
[1]  7  9 11 13 15

> mm$dif
[1] 5 5 5 5 5
```

Сначала мы создали функцию для нахождения разности и суммы двух векторов. Эта функция возвращает список из двух объектов, представляющих собой два вектора — результат сложения и вычитания двух исходных векторов. Далее создали векторы `x1` и `x2`. Подставили их в качестве аргументов в функцию `ff`. Сохранили результат в переменную `mm`. Вывели результат на консоль.

Операторы цикла и условия

При написании функций и обработке данных часто требуется использование операторов цикла и условия.

В R можно выделить два оператора условия: **`if`** и **`ifelse`**. Полный вариант оператора `if` записывается в следующем виде:

```
if(условие) {выражение1} else {выражение2}
```

Здесь «условие» — это любое логическое выражение с операторами условия (>, <, ==, >=, <=, !=). Если результат сравнения будет «истина», то выполняется блок команд 1, если «ложь», то блок команд 2. Например:

```
> x<-16
> if(x>0) {y<-sqrt(x)} else {y<-0}
> y
[1] 4
```

При этом присутствие **`else`** в выражении не обязательно.

Особенностью оператора `if` является то, что сравнивать можно только векторы единичной длины, то есть скаляры. Для того чтобы сравнить два вектора или вектор со скаляром, нужно использовать оператор `ifelse`. Он записывается как:

```
ifelse(условие, yes, no)
```

и позволяет переменной в зависимости от выполнения условия принимать различные значения.

В примере ниже в новый вектор записываются совпадающие элементы двух исходных векторов, а если соответствующие элементы не совпадают, то записывается 0.

```
> x<-c(1, 3, 7, 10, 12, 14)
> y<-c(2, 3, 8, 10, 11, 14)
> ifelse(x==y, x, 0)
[1]  0  3  0 10  0 14
```

В R существует несколько операторов цикла **for**, **while** и **repeat**. Формы записи у них следующие:

```
for(переменная_in_последовательность целых чисел) {блок
    команд}

while(условие) {блок команд}

repeat {блок команд}
```

Циклы `for`: команда или блок команд выполняется до тех пор, пока переменная находится в рамках заданной числовой (или символьной) последовательности. В качестве примера рассмотрим вычисление факториала n!:

```
> n<-4
> x<-1
> for(i in 1:n) {x<-x*i}
> x
[1] 24
```

Циклы `while`: блок команд выполняется пока справедливо условие, например, переменная меньше определённого значения:

```
> x<-(-10)
> while(x<0) {x<-x+1}
> x
[1] 0
```

В случае, если применяется оператор `repeat`, — создаётся бесконечный цикл, для выхода из которого нужно использовать команду **break** в сочетании с оператором условия `if` в блоке команд:

```
> x<-(-10)
> repeat {if(x<0) {x<-x+1} else {break}}
> x
[1] 0
```

Функции apply(), lapply() и sapply()

Важной особенностью R является возможность векторизации вычислений. Допустим, необходимо найти натуральный логарифм для каждого элемента вектора `x`. Вместо цикла, где функция `log(x)` применяется поочерёдно к каждому элементу вектора, подставим вектор целиком в функцию `log()`, которая вычисляет логарифм для каждого элемента:

```
> x<-1:5
> log(x)
[1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379
```

Принцип векторизации применим и к более сложным объектам, таким как матрицы, таблицы данных и списки.

Функция **`apply()`** используется в тех случаях, когда необходимо применить какую-либо функцию ко всем строкам или столбцам матрицы. Предположим, что нам нужно найти суммы элементов матрицы на каждой строке, используя функцию **`sum()`**. С этой целью можно было бы использовать цикл, но гораздо проще использовать функцию `apply()`:

```
> mm<-matrix(1:9, ncol=3)
> mm
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

> sm<-apply(mm, 1, sum)
> sm
[1] 12 15 18
```

Параметр, равный 1, означает, что функция `sum()` будет применена к **строкам**, а если параметр равен 2, то функция будет применена к **столбцам**. Результатом вычислений будет вектор суммарных значений.

Функции **lapply()** и **sapply()** используются, когда нужно применить какую-либо функцию к компонентам списка. Функция `lapply()` возвращает результат также в виде списка, а `sapply()` — в виде вектора.

Посчитаем средние значения для каждого из трёх векторов, являющихся компонентами списка `xx`:

```
> xx<-list(a=1:5, b=10:15, c=30:100)
> yy<-sapply(xx, mean)
> yy
  a     b     c
3.0  12.5  65.0
```

Практическое задание

1. Импортируйте и сохраните в переменную содержимое файла «**Response2Drug.txt**». Используя оператор `ifelse`, замените столбец `age`, который содержит числовые значения возраста, на столбец, содержащий категории возраста: «`young`», «`middle`», «`old`». Сохраните полученную таблицу в файл.

2. Напишите функцию для расчёта двойного факториала числа. Учтите, что аргументом этой функции могут быть только целые неотрицательные числа, не равные нулю. Для того чтобы определить делится ли число на 2, используйте оператор %% (остаток от деления). Например, 4%%2 даст результат 0. Сделайте так, чтобы для нуля и отрицательных значений эта функция возвращала NA. Используя полученную функцию, рассчитайте значения двойного факториала для 6, 7, 0 и -2.

3. Создайте вектор случайных чисел от –10 до 50 длиной 25. Используйте для этого функцию `runif()`. Создайте на основе полученного вектора квадратную матрицу. Посчитайте средние значения и суммы элементов для строк и столбцов матрицы.

4. Напишите функцию `logint(a, b, base=2)` для расчёта интеграла от логарифма по основанию `base` в пределах от `a` до `b`. По умолчанию основание логарифма равно 2. Для расчёта интеграла используйте метод прямоугольников:

$$\int_a^b f(x)dx \approx \sum_{i=1}^{n} f\left(\frac{x_1 + x_{i+1}}{2}\right)(x_{i+1} - x_i),$$

где n — число интервалов. Учтите, что нижний предел для логарифма должен быть больше нуля.

Используя написанную функцию, посчитайте значение интеграла для натурального логарифма в пределах от 2 до 4. Равно ли полученное значение 2,159? Примечание: сделайте параметр `base=exp(1)`, чтобы основание логарифма было равно числу e.

Вопросы

1. Какие Вы знаете операторы условия?
2. Какие Вы знаете операторы цикла?
3. Для чего нужны функции `apply()`, `lapply()` и `sapply()`?

ТЕМА 4.

БАЗОВАЯ ГРАФИКА. ПОНЯТИЕ О ФУНКЦИЯХ ВЫСОКОГО И НИЗКОГО УРОВНЯ

В R все функции, отвечающие за графики, можно разделить, по меньшей мере, на два типа: функции **высокого** уровня и функции **низкого** уровня. Функции высокого уровня приводят к созданию графического окна, в котором строится изображение. В свою очередь, функции низкого уровня не могут создавать графическое окно, а лишь добавляют новые элементы на уже существующий график.

Рисование графиков: функции высокого уровня

- `plot(x)` — график `x`
- `plot(x, y)` — график зависимости `y` от `x`
- `hist(x)` — гистограмма
- `barplot(x)` — столбчатая диаграмма
- `dotchart(x)` — диаграмма Кливленда
- `pie(x)` — круговая диаграмма
- `boxplot(x)` — график типа «ящика с усами»
- `sunflowerplot(x, y)` — то же, что и `plot()`, однако точки с одинаковыми координатами изображаются в виде «ромашек», количество лепестков у которых пропорционально количеству таких точек
- `coplot(x~y | z)` — график зависимости `y` от `x` для каждого интервала значений `z`
- `interaction.plot(f1, f2, y)` — если `f1` и `f2` являются факторами, эта функция создаст график со средними значениями `y` в соответствии со значениями `f1` (по оси X) и `f2` (по оси У, разные кривые)
- `matplot(x, y)` — график зависимости столбцов `y` от столбцов `x`

- `fourfoldplot(x)` — изображает (в виде частей окружности) связь между двумя бинарными переменными в разных совокупностях
- `assocplot(x)` — график Кохена—Френдли
- `mosaicplot(x)` — мозаичный график остатков лог-линейной регрессии
- `pairs(x)` — если x является матрицей или таблицей данных, то функция изображает диаграммы рассеяния для всех возможных пар переменных из x
- `plot.ts(x)`, `ts.plot(x)` — изображает временной ряд
- `qqnorm(x)` — диаграмма квантилей
- `qqplot(x, y)` — график зависимости квантилей `y` от квантилей x
- `contour(x, y, z)` — выполняет интерполяцию данных и создаёт контурный график
- `filled.contour(x, y, z)` — то же, что `contour()`, но заполняет области между контурами определёнными цветами
- `image(x, y, z)` — изображает исходные данные в виде квадратов, цвет которых определяется значениями x и y
- `persp(x, y, z)` — то же, что и `image()`, но в виде трёхмерного графика
- `stars(x)` — если `x` является матрицей или таблицей данных, то функция изображает график в виде «звёзд» так, что каждая строка представлена «звездой», а столбцы задают длину сегментов этих «звёзд»
- `symbols(x, y, ...)` — изображает различные символы в соответствии с координатами
- `termplot(mod.obj)` — изображает частные эффекты переменных из регрессионной модели

Рисование графиков: функции низкого уровня

- `points(x, y)` — рисование точек
- `lines(x, y)` — рисование линии
- `text(x, y, labels, ...)` — добавление текстовой надписи в заданных координатах
- `mtext(text, side=3, line=0, ...)` — добавление текстовой надписи к одному из полей графика

- `segments(x0, y0, x1, y1)` — рисование отрезка
- `arrows(x0, y0, x1, y1, angle=30, code=2)` — рисование стрелки
- `abline(a,b)` — рисование наклонной прямой
- `abline(h=y)` — рисование вертикальной прямой
- `abline(v=x)` — рисование горизонтальной прямой
- `abline(lm.obj)` — рисование регрессионной прямой
- `rect(x1, y1, x2, y2)` — рисование прямоугольника
- `polygon(x, y)` — рисование многоугольника
- `legend(x, y, legend)` — добавление легенды
- `title()` — добавление заголовка
- `axis(side, vect)` — добавление осей
- `rug(x)` — рисование засечек на оси X
- `locator(n, type="n", ...)` — возвращает координаты на графике, на которые кликнул пользователь

Функция высокого уровня plot()

Основной функцией для построения двумерных графиков является функция **`plot(x, y, ...)`**. Эта функция позволяет строить зависимость `y` от `x`. Основные аргументы функции описаны в таблице 4.

Таблица 4. Аргументы функции `plot()`

Аргумент	Описание
`x`	Координаты точки по оси X
`y`	Координаты точки по оси Y
`type`	Символьный аргумент, определяющий тип графика. Некоторые значения: «p» — точки, «l» — линии, «b» — линии и точки
`lty`	Стиль линии. Некоторые значения: 1 — сплошная линия, 2 — штриховая линии, 3 — пунктирная линия, 4 — штрих-пунктир
`xlab`, `ylab`, `main`, `sub`	Названия осей, основного и дополнительного заголовков
`col`	Цвет графика (цвет точек и соединяющих линий). Цвета, доступные в R, можно просмотреть при помощи команды `colors()`

Окончание таблицы 4.

Аргумент	Описание
`col.axis`, `col.main`, `col.sub`	Цвет осей, основного и дополнительного заголовков
`bg`, `fg`	Цвет фона окна и цвет рамки вокруг графика
`font.axis`, `font.main`, `font.sub`, `font.lab`	Тип шрифта для названия осей, основного и дополнительного заголовков, подписей к осям. Значения: 1 — простой, 2 — жирный, 3 — курсив, 4 — жирный курсив
`lwd`	Толщина линии по отношению к 1 (100 %)
`cex`	Размер точек на графике по отношению к 1 (100 %)
`cex.axis`, `cex.main`, `cex.sub`, `cex.lab`	Размер значений на осях, размер заголовка, дополнительного заголовка, подписей к осям. Размер по умолчанию 1 (100 %)
`pch`	Тип рисуемого символа с возможными значениями: от 0 до 255. Примеры символов: 0 — □, 1 — ○, 2 — Δ, 3 — +, 4 — ×

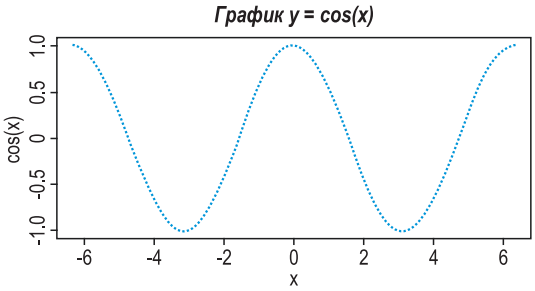


Рис. 2. График $y = \cos(x)$

В качестве примера создадим график функции:

$$y = \cos(x).$$

Сделаем так, что график будет изображён пунктирной линией синего цвета, название графика зададим жирным курсивом (рис. 2).

```
> x<-seq(-2*pi, 2*pi, by=0.01)
> y<-cos(x)
> plot(x, y, type="l", lty=3, col="blue", main="График
    y=cos(x)", font.main=4, ylab="cos(x)")
```

Функции низкого уровня abline(), lines(), points(), legend() и text()

Рассмотрим некоторые функции низкого уровня.

- Функция **`abline()`** добавляет на график прямую линию. Линию можно задать двумя способами: 1) указать параметры `h` или `v`, представляющие собой точки пересечения осей Y и X; 2) указать коэффициенты a и b из уравнения $y = ax + b$.
- Функция **`lines()`** добавляет на график точки, соединённые отрезками. Для этого нужно указать координаты точек `x` и `y`, таким образом, функция принимает вид: `lines(x, y)`.
- Функция **`points()`** добавляет на график точки. Для этого нужно указать координаты точек `x` и `y`, функция принимает вид: `points(x, y)`.

Для этих функций также доступны аргументы, определяющие цвет, толщину и тип линий (`col`, `lwd`, `lty`), для функций `lines()` и `points()` — тип символов (`pch`).

В качестве примера добавим на уже построенный график функции $y = \cos(x)$ второй график — $y = \sin(x)$. Для визуализации графика второй функции используем красный цвет, а треугольники будут соединены сплошной линией; добавим к графику прямую штриховую линию, проходящую через точку 0 оси Y (рис. 3).

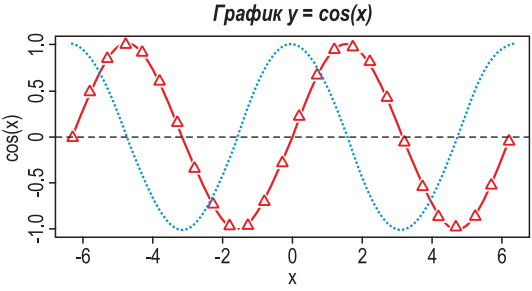


Рис. 3. Графики $y = \cos(x)$ (синяя линия) и $y = \sin(x)$ (красная линия)

```
> x<-seq(-2*pi, 2*pi, by=0.01)
> y<-cos(x)
> plot(x, y, type="l", lty=3, col="blue",
    main="График y=cos(x)", font.main=4, ylab="cos(x)")
> x<-seq(-2*pi, 2*pi, by=0.5)
> y<-sin(x)
> lines(x, y, type="b", lty=1, pch=2, col="red")
> abline(h=0, lty=2)
```

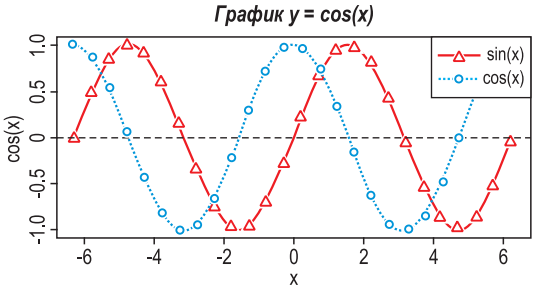


Рис. 4. Графики $y = \cos(x)$ (синяя линия) и $y = \sin(x)$ (красная линия) с добавленной в правом верхнем углу легендой

Теперь, когда на одном графике представлены несколько функций, необходимо добавить легенду с указанием расшифровки обозначений. Это можно сделать с помощью функции `legend()` (рис. 4).

```
> x<-seq(-2*pi,2*pi,by=0.5)
> y<-cos(x)
> plot(x, y, type="b", lty=3, col="blue", main="График y=cos(x)",
     font.main=4, pch=1, ylab="cos(x)")
> x<-seq(-2*pi, 2*pi, by=0.5)
> y<-sin(x)
> lines(x, y, type="b", lty=1, pch=2, col="red")
> abline(h=0, lty=2)
> legend("topright", legend=c("sin(x)", "cos(x)"), lty=c(1, 3),
     col=c("red", "blue"), pch=c(2, 1))
```

Первый аргумент функции определяет местоположение легенды (`topright` — в верхнем правом углу), второй аргумент — текст легенды, а последующие — цвет, тип символа и прочее. Также можно указать конкретные координаты легенды на графике при помощи мыши. Для этого нужно запустить функцию **`locator(1)`**, навести курсор на нужное место графика и нажать левую кнопку мыши. Координаты будут записаны в переменную типа «список»:

```
> xy<-locator(1)
> xy
$x
[1] -5.986454
$y
[1] -0.4611233
```

Полученные координаты можно подставить в функцию `legend()`:

```
> legend(xy, legend=c("sin(x)", "cos(x)"), lty=c(1, 3),
      col=c("red", "blue"), pch=c(2, 1))
```

Функция `text()` добавляет на график текст в соответствии с указанными координатами. Построим график из трёх точек и добавим подписи к каждой из них (рис. 5):

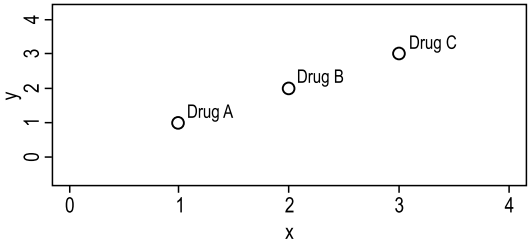


Рис. 5. Три точки с подписями к каждой из них

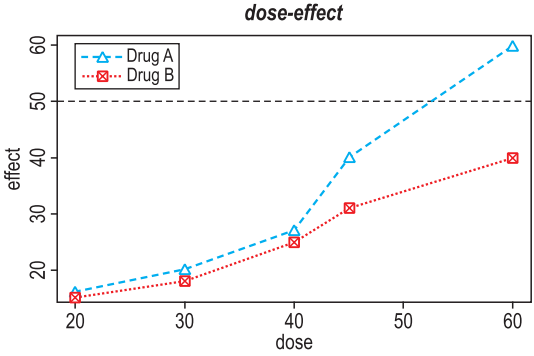
```
> x<-c(1, 2, 3)
> y<-c(1, 2, 3)
> plot(x, y, type="p", xlim=c(0, 4), ylim=c(0, 4), cex=1.5)
> lb<-c("Drug A", "Drug B", "Drug C")
> text(x+0.3, y+0.3, labels=lb)
```

Практическое задание

1. Дана информация об эффектах двух лекарств (в процентах) в зависимости от дозы:

Dose	Drug A	Drug B
20	16	15
30	20	18
40	27	25
45	40	31
60	60	40

Используя функцию `plot()`, а также различные функции низкого уровня, постройте диаграмму как на примере ниже.



Вопросы

1. Что такое функции высокого и низкого уровня для построения графиков в R?
2. Какие Вы знаете функции низкого уровня?

ТЕМА 5.

РАЗЛИЧНЫЕ ВИДЫ ДИАГРАММ. ГИСТОГРАММЫ, ДИАГРАММЫ РАЗМАХОВ. СТОЛБЧАТЫЕ И КРУГОВЫЕ ДИАГРАММЫ

Помимо функции `plot()` в R есть и другие функции высокого уровня. Они позволяют строить диаграммы размахов, гистограммы, столбчатые и круговые диаграммы.

Диаграммы размахов, гистограммы

Диаграмма размахов («ящик с усами») иллюстрирует распределение непрерывной величины, отображая описательную статистику: максимальное и минимальное значение, медиану, верхний и нижний квартили. Диаграмма также иллюстрирует наличие выбросов — объектов, значения которых выходят за пределы ±1,5 межквартильного размаха. Межквартильный размах — это разность между верхним и нижним квартилями. Сравнение плотности распределения и «ящика с усами» показано на рисунке 6.

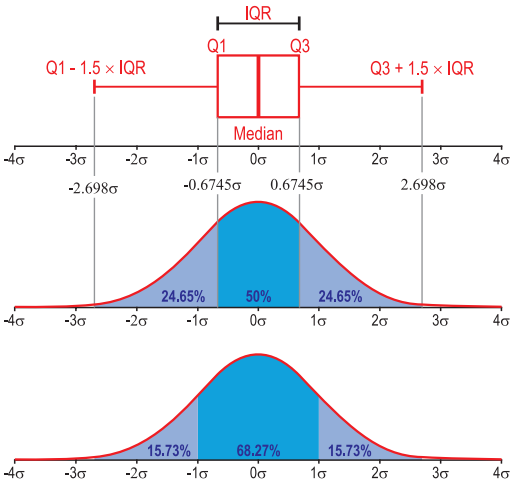


Рис. 6. Сравнение плотности распределения и «ящика с усами»

Построить диаграмму размахов позволяет функция **`boxplot()`**.

Рассмотрим особенности её использования на конкретном примере.

Загрузим существующий в R набор данных «ToothGrowth» при помощи команды **data()**. Это данные о длине зубов морских свинок, которые получали апельсиновый сок или витамин С. Построим диаграмму размахов для длины зубов (рис. 7):

```
> data(ToothGrowth)
> head(ToothGrowth)
   len supp dose
1  4.2   VC  0.5
2 11.5   VC  0.5
3  7.3   VC  0.5
4  5.8   VC  0.5
5  6.4   VC  0.5
6 10.0   VC  0.5
```

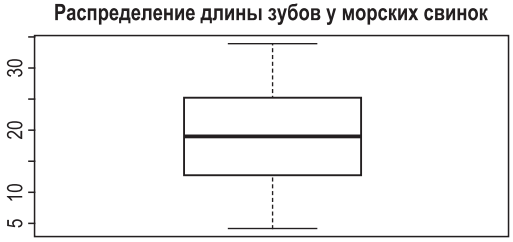


Рис. 7. Распределение длины зубов морских свинок

```
> boxplot(ToothGrowth$len, main="Распределение длины
    зубов у морских свинок")
```

Для того чтобы построить два «ящика с усами» на одной диаграмме, которые будут описывать распределение длины зубов для свинок, получавших апельсиновый сок и витамин С, в качестве первого аргумента функции нужно указать формулу **«название столбца 1» ~ «название столбца 2»**. В качестве значения параметра `data` необходимо указать название таблицы, где содержатся эти столбцы. Аргументам **varwidth** и **notch** присвоим значение «истина»: параметр `varwidth=TRUE` позволяет получить диаграмму, на которой ширина «ящиков» будет пропорциональна квадратному корню из объёма выборки, тогда как `notch=TRUE` добавляет к «ящикам» насечки, ширина которых пропорциональна ширине 95% доверительного интервала для медианы. Если насечки не перекрываются, то велика вероятность того, что медианы соответствующих генеральных совокупностей также различаются. Кроме того, используя вектор `c("green", "red")`, «ящикам» присваиваются разные цвета (рис. 8).

```
> boxplot(len~supp, data=ToothGrowth, varwidth=TRUE,
    notch=TRUE, col=c("green", "red"), main="Распределение
    длины зубов у морских свинок")
```

Из рисунка 8 видно, что насечки не перекрываются, следовательно, можно сделать вывод, что апельсиновый сок (OJ) сильнее влияет на рост зубов, по сравнению с витамином С (VC).

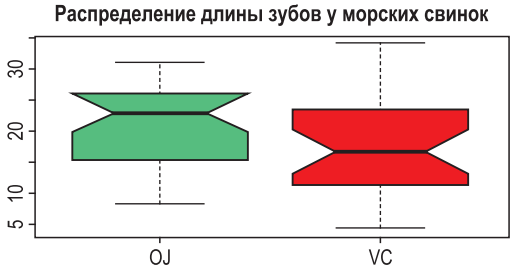


Рис. 8. Распределение длины зубов морских свинок, получавших апельсиновый сок (OJ) и витамин С (VC):
Ширина насечек на «ящиках» пропорциональна ширине 95% доверительного интервала для медианы

Гистограммы также описывают распределение случайной величины. Построить гистограмму можно при помощи функции **`hist(x)`**, где `x` — числовой вектор. Построим гистограмму для длины зубов морских свинок (рис. 9).

```
> hist(ToothGrowth$len, col="lightblue")
```

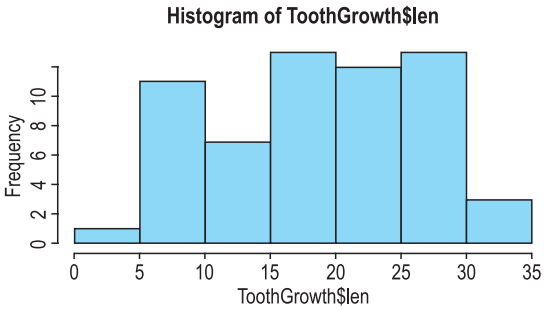


Рис. 9. Гистограмма распределения длины зубов морских свинок

На гистограмме по оси Y указано количество свинок, длина зубов которых попадает в данный интервал.

Добавим к гистограмме график плотности вероятности при помощи функции низкого уровня `lines()`, вычислив плотность вероятности при помощи функции `density()` (рис. 10). Если установить аргумент **`freq=FALSE`**, то на оси Y будет указана плотность вероятности.

```
> hist(ToothGrowth$len, col="lightblue", freq=FALSE,
    ylim=c(0, 0.05))
> lines(density(ToothGrowth$len), col="red", lwd=2)
```

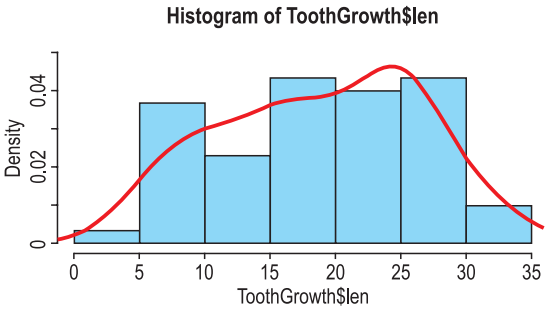


Рис. 10. Гистограмма распределения длины зубов морских свинок: Красная линия показывает плотность вероятности

При оформлении результатов анализа данных часто бывает необходимо объединить несколько диаграмм в одну. Этого можно добиться, воспользовавшись функцией **`par()`**. Она позволяет устанавливать графические параметры для всех рисунков, которые строит пользователь во время R-сессии, например `col`, `pch`, `cex` и многие другие. Эти же параметры можно установить для каждого рисунка по отдельности, как это было показано во всех предыдущих примерах.

Разместим все четыре рисунка на одной диаграмме, заполнив её построчно, указав параметр `mfrow=c(2, 2)` функции `par()` (рис. 11). Однако сначала сохраним текущие значения графических параметров, чтобы потом можно было легко вернуться к значениям по умолчанию.

```
> pr<-par(no.readonly=TRUE)
> par(mfrow=c(2, 2))
> boxplot(ToothGrowth$len, main="Длина зубов")
> boxplot(len~supp, data=ToothGrowth, varwidth=TRUE,
    notch=TRUE, col=c("green", "red"), main="Длина зубов")
> hist(ToothGrowth$len, col="lightblue", main="Длина зубов")
> hist(ToothGrowth$len, col="lightblue", freq=FALSE,
    main="Длина зубов")
> lines(density(ToothGrowth$len), col="red", lwd=2)
> par(pr)
```

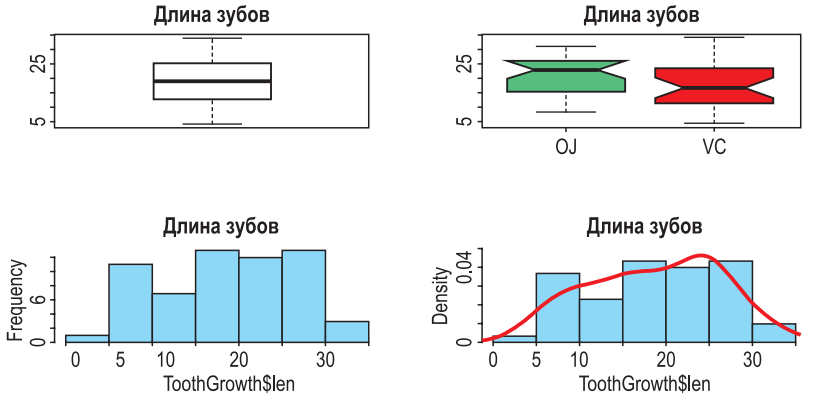


Рис. 11. Гистограммы распределения (два нижних фрагмента) и диаграммы размахов (два верхних фрагмента) длины зубов морских свинок, расположенные на одном графике:

Свинки получали апельсиновый сок (OJ) и витамин C (VC). Красная линия показывает плотность вероятности

Для того чтобы построить график, где на верхней строке будет одна диаграмма, а на нижней строке две, следует воспользоваться функцией **`layout()`**. Сначала создадим матрицу, описывающую то, как будут располагаться диаграммы:

```
> mm<-matrix(c(1, 1, 2, 3), ncol=2, nrow=2, byrow=T)
> mm
     [,1] [,2]
[1,]    1    1
[2,]    2    3
```

Далее подставляем её в функцию `layout()` и строим графики (рис. 12):

```
> layout(mm)
> hist(ToothGrowth$len, col="lightblue", main="Длина зубов")
> boxplot(len~supp, data=ToothGrowth, varwidth=TRUE,
    notch=TRUE, col=c("green", "red"), main="Длина зубов")
> hist(ToothGrowth$len, col="lightblue", freq=FALSE,
    main="Длина зубов")
> lines(density(ToothGrowth$len), col="red", lwd=2)
```

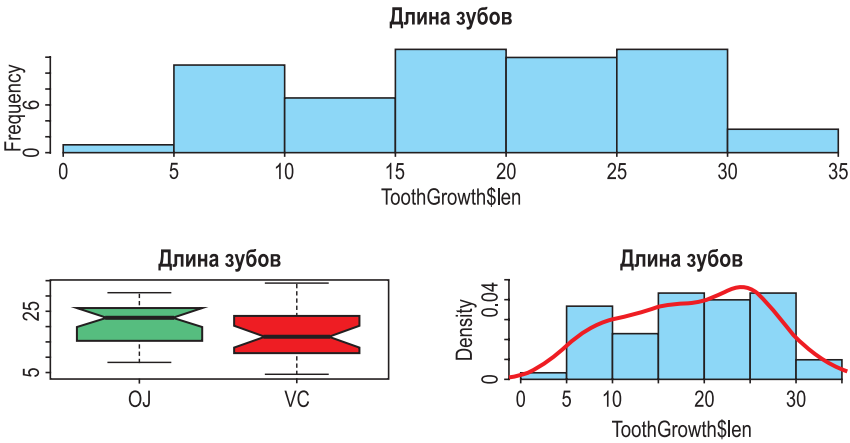


Рис. 12. Гистограммы распределения и диаграмма размахов (левый нижний фрагмент) длины зубов морских свинок, расположенные на одном графике (1:2):

Свинки получали апельсиновый сок (OJ) и витамин C (VC). Красная линия показывает плотность вероятности

Столбчатые и круговые диаграммы

Столбчатые диаграммы отражают распределение значений категориальной переменной. Различают *простые* столбчатые диаграммы, когда анализу подвергается одна категориальная переменная, а также *составные* диаграммы и диаграммы *с груп-*

пировкой, когда проводится анализ связи двух категориальных переменных. Построить столбчатые диаграммы можно при помощи функции **`barplot()`**. Рассмотрим особенности использования этой функции на примере данных по степени гепатотоксичности лекарственных веществ. Загрузим соответствующие данные из файла «**Hepatotoxicity.txt**»:

```
> tt<-read.delim("Hepatotoxicity.txt", as.is=T)
> head(tt)
          Generic.name        Class     Dosage
1              Abacavir     Moderate       High
2        Acetaminophen     Moderate       High
3             Aciclovir      Without       High
4   Adefovir Dipivoxil      Without        Low
5             Alfuzosin     Moderate        Low
6           Allopurinol       Severe       High
```

Соответствующая таблица содержит несколько столбцов: название лекарства, класс гепатотоксичности (выраженная, умеренная, отсутствие) и доза (низкая, средняя, высокая). Построим простую столбчатую диаграмму для класса гепатотоксичности. Посчитать количество лекарств каждого класса гепатотоксичности можно при помощи функции `table()`:

```
> counts<-table(tt$Class)
> counts
Moderate    Severe   Without
      66        47        38
```

Если подставить полученную таблицу в качестве аргумента функции `barplot()`, то можно получить следующий график (рис. 13):

```
> barplot(counts, col=c("red", "blue", "green"), cex.names=2,
    cex.main=2, main="Распределение классов
    гепатотоксичности")
```

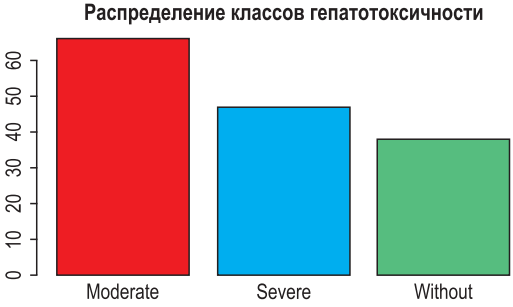


Рис. 13. Столбчатая диаграмма распределения класса гепатотоксичности лекарственных веществ

Для того чтобы исследовать связь между степенью гепатотоксичности и дозой лекарства, построим составную столбчатую диаграмму или столбчатую диаграмму с группировкой. Чтобы это сделать, рассчитаем таблицу сопряжённости для двух переменных и подставим её в качестве аргумента функции `barplot()`. *Таблица сопряжённости* рассчитывается при помощи функции `table()`:

```
> counts<-table(tt$Class, tt$Dosage)
> counts
          High  Low  Medium
Moderate    35    9      22
Severe      36    4       7
Without      8   19      11

> counts<-counts[c(2, 1, 3), c(1, 3, 2)]
> counts
          High  Medium  Low
Severe      36       7    4
Moderate    35      22    9
Without      8      11   19
```

Помещаем таблицу сопряжённости в переменную `counts`, а потом меняем местами строки и столбцы, чтобы степени гепатотоксичности и дозы шли по порядку. Далее строим составную столбчатую диаграмму (рис. 14):

```
> barplot(counts, col=c("red", "blue", "green"), cex.names=2,
    cex.main=2, legend=rownames(counts), main="Зависимость
    степени гепатотоксичности от дозы")
```

Из диаграммы на рисунке 14 видно, что с увеличением дозы степень гепатотоксичности увеличивается.

Для визуализации данных в виде столбчатой диаграммы с группировкой нужно задать аргумент **beside=TRUE** (рис. 15):

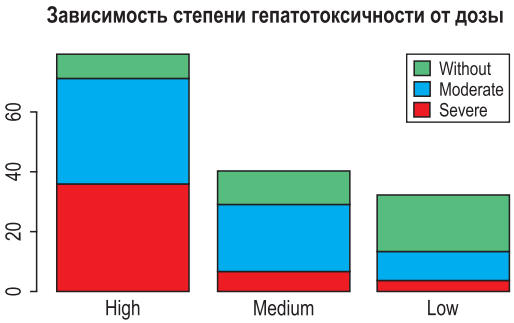


Рис. 14. Составная столбчатая диаграмма зависимости степени гепатотоксичности лекарственных веществ от дозы

```
> barplot(counts, col=c("red", "blue", "green"), cex.names=2,
    cex.main=2, legend=rownames(counts), main="Зависимость
    степени гепатотоксичности от дозы", beside=TRUE)
```

Круговую диаграмму можно построить с помощью функции **pie()**. Построим соответствующую диаграмму для степеней гепатотоксичности (рис. 16):

```
> counts<-table(tt$Class)
> pie(counts, labels=names(counts), col=c("red", "blue",
    "green"), cex=2)
```

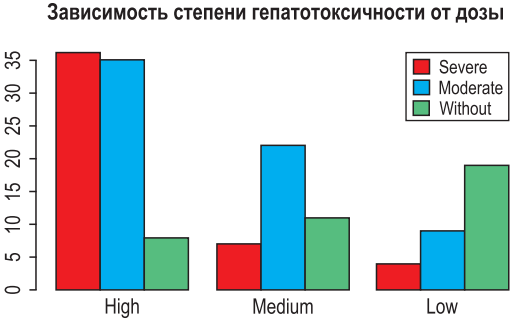


Рис. 15. Столбчатая диаграмма с группировкой зависимости степени гепатотоксичности лекарственных веществ от дозы

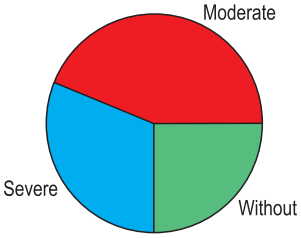


Рис. 16. Круговая диаграмма степеней гепатотоксичности лекарственных веществ

Практическое задание

1. В таблице из файла «**Arthritis.txt**» представлены данные по эффекту нового лекарства для лечения ревматоидного артрита. Терапевтический эффект подразделяется на три категории: `market` — состояние пациента улучшилось значительно, `some` — улучшилось незначительно, `none` — состояние не изменилось. Создайте простую столбчатую диаграмму для терапевтического эффекта. Добавьте заголовок, подписи осей. Для каждого столбца укажите свой цвет. Подберите подходящий размер значений на осях, подписей, заголовка, их шрифт и цвет.

Создайте соответствующую круговую диаграмму. Сделайте так, чтобы подписи сегментов содержали процент от числа пациентов. Для этого сначала посчитайте проценты, а потом используйте функцию paste() для того, чтобы склеить проценты с названиями сегментов, например: `paste("названия сегментов", " " , "проценты", "%", sep=" ")`.

Постройте составную столбчатую диаграмму и диаграмму с группировкой, чтобы на них была показана группа пациентов (пациенты, принимающие лекарство или плацебо) и степень терапевтического эффекта. Подберите для этих диаграмм подходящие параметры (заголовки, подписи осей, цвет и т. д.). Добавьте легенды. Объедините две диаграммы в одну. Есть ли связь между приёмом лекарства и наблюдаемым эффектом?

Сохраните все построенные диаграммы в виде файлов с расширениями .jpeg или .tiff. Сделать это можно при помощи «Export/Save as Image» в нижнем правом окне RStudio.

2. В таблице из файла «**mRNA-protein correlation.txt**» представлены концентрации матричной РНК и белка для 4962 генов мыши. Концентрации измерены в мышиных фибробластах и представлены в виде числа молекул на клетку. Импортируйте и сохраните таблицу в переменную. Переименуйте названия столбцов, сделав их более короткими, чтобы потом можно было обращаться к столбцам по имени. Проверьте, есть ли пропущенные значения концентраций РНК или белка. Если есть, то удалите соответствующие строки. Сколько генов (строк) осталось в таблице?

Постройте гистограммы и диаграммы размахов для РНК и белка. Информативны ли соответствующие графики?

Преобразуйте исходные значения концентраций в логарифмы. Постройте снова те же самые диаграммы. Изменились ли они? Какие выводы можно сделать исходя из них? Объедините их в одну диаграмму, так чтобы на верхней строке были две гистограммы, а на нижней — диаграмма размахов. Для этого используйте функцию `layout()`. Не забудьте сначала сохранить в переменную текущие значения параметров, например `opar<-par(no.readonly=T)`, чтобы потом можно было восстановить исходные значения.

Постройте диаграмму зависимости логарифмов концентраций белка от мРНК, используя функцию `plot()`. Подберите подходящие параметры (заголовки, цвет, размер точек и др.). Зависит ли концентрация белка в клетке от концентрации соответствующей мРНК? Посчитайте коэффициент корреляции Пирсона с помощью функции `cor()`. О чём говорят полученные значения? Можете ли Вы объяснить полученный результат? Добавьте значения коэффициента корреляции Пирсона на график (выражение вида «R = значение»). Используйте функцию `text()` для добавления текста и `locator(1)`, чтобы определить координаты вставки.

Сохраните все построенные диаграммы в виде файлов с расширениями .jpeg или .tiff.

Вопросы

1. Какие функции высокого уровня есть в R?
2. Какие функции высокого уровня используются для анализа числовых данных?
3. Какие функции высокого уровня используются для анализа категориальных данных?

РЕКОМЕНДУЕМАЯ ЛИТЕРАТУРА

Основная

- *Зарядов И.С.* Введение в статистический пакет R: типы переменных, структуры данных, чтение и запись информации, графика. Москва, 2010.
- *Зарядов И.С.* Статистический пакет R: теория вероятностей и математическая статистика. Москва, 2010.
- *Мастицкий С.Э., Шитиков В.К.* Статистический анализ и визуализация данных с помощью R. Москва, 2014.
- *Шипунов А.Б., Балдин Е.М., Волкова П.А. и др.* Наглядная статистика. Используем R! Москва, 2014.
- *Буховец А.Г.* Алгоритмы вычислительной статистики в системе R. Санкт-Петербург, 2015.
- *Шитиков В.К., Мастицкий С.Э.* Классификация, регрессия и другие алгоритмы Data Mining с использованием R [Электронный ресурс]. URL: https://github.com/ranalytics/data-mining/blob/master/Book/Shitikov_Mastitsky_2017_DM_R.pdf

Дополнительная

- *Кабаков Р.И.* R в действии. Анализ и визуализация данных на языке R. Москва, 2014.

Для заметок

Учебное издание

ВВЕДЕНИЕ В СТАТИСТИЧЕСКИЙ ЯЗЫК ПРОГРАММИРОВАНИЯ R

Учебно-методическое пособие

Составители:
Иванов Сергей Михайлович,
Муравьева Елена Степановна,
Лагунин Алексей Александрович

Редактор, оформление Г.В. Нестерова
Корректор Е.Н. Королева
Художественный редактор Т.В. Нестерова

Подписано в печать 25.09.2025.
Формат $60 \times 90^{1}/_{16}$. Усл. печ. л. 4. Тираж 100 экз. Заказ № 44-25.

Отпечатано в ФГАОУ ВО РНИМУ им. Н.И. Пирогова
Минздрава России (Пироговский Университет)
117513, Москва, ул. Островитянова, 1
www.rsmu.ru

ISBN 978-5-88458-770-0

