

| | |
|---------------|---|
| ДИСЦИПЛИНА | Языки программирования для статистической обработки данных |
| | _____ |
| | полное название дисциплины без аббревиатуры |
| ИНСТИТУТ | ИИТ |
| КАФЕДРА | Прикладная математика |
| | _____ |
| | полное название кафедры |
| ГРУППА/Ы | ИМБО-01,02-20; ИНБО-05,06,08,10-20 |
| | _____ |
| | номер групп/ы, для которых предназначены материалы |
| ВИД УЧЕБНОГО | Практикум |
| МАТЕРИАЛА | (лекция; материал к практическим занятиям; контрольно-измерительные материалы к практическим занятиям; руководство к КР/КП, практикам |
| ПРЕПОДАВАТЕЛЬ | Митина Ольга Алексеевна |
| | _____ |
| | фамилия, имя, отчество |
| СЕМЕСТР | 4 |
| | _____ |
| | указать номер семестра обучения |

Оглавление

| | |
|---|-----|
| Практическая работа № 1 | 4 |
| Назначение и основные возможности R | 4 |
| Векторы..... | 17 |
| Самостоятельная работа №1 | 28 |
| Практическая работа № 2 | 34 |
| Матрицы | 34 |
| Операции над матрицами..... | 36 |
| Самостоятельная работа №2..... | 43 |
| Практическая работа №3 | 49 |
| Создание и обработка фреймов..... | 49 |
| Работа с датафреймами с использованием библиотеки dplyr | 51 |
| Экспорт данных из R в другие форматы | 53 |
| Списки (lists) | 57 |
| Преобразование типов и структур данных | 61 |
| Самостоятельная работа №3..... | 61 |
| Практическая работа №4 | 66 |
| Ввод-вывод данных | 66 |
| Импорт данных из текстовых файлов | 67 |
| Применение буфера обмена при импорте данных | 70 |
| Импорт из EXCEL-файлов..... | 71 |
| Импорт из файлов *.csv..... | 75 |
| Практическая работа №5 | 80 |
| Программирование разветвлений и циклов..... | 80 |
| Организация циклов | 82 |
| Самостоятельная работа №5..... | 85 |
| Практическая работа №6 | 92 |
| Создание пользовательских функций..... | 92 |
| Самостоятельная работа..... | 96 |
| Практическая работа №7 | 101 |
| Двухмерная графика..... | 101 |
| Трёхмерные графики и линии уровня | 103 |
| Самостоятельная работа..... | 108 |
| Практическая работа №8 | 110 |
| Некоторые сведения о возможностях статистического анализа | 110 |
| Гистограмма и эмпирическая плотность..... | 113 |
| Эмпирическая функция распределения | 114 |

| | |
|---|-----|
| Самостоятельная работа №8..... | 115 |
| Практическая работа №9 | 117 |
| Тестирование в testthat | 117 |
| Самостоятельная работа №9..... | 118 |
| Практическая работа №10 | 119 |
| Построение графики в пакете ggplot2 | 119 |
| Самостоятельная работа №10..... | 127 |
| Практическая работа №11 | 131 |
| ООП в R и S3 классы..... | 131 |
| Список использованной литературы..... | 140 |

Практическая работа № 1

Назначение и основные возможности R

R – язык программирования и программная среда для математического моделирования, выполнения статистических расчетов и графического анализа сложных прикладных процессов из различных областей деятельности, включая финансы, экономику, производство, менеджмент, страхование, социологию и др.

Создателями R являются Росс Айхека (Ross Ihaka) и Роберт Джентелмен (Robert Gentleman), представившие свой продукт в новозеландском университете Окленда в 1993 г. как свободное, бесплатно распространяемое, с открытым кодом программное обеспечение. Благодаря этим особенностям, R быстро и постоянно расширяет свои возможности – над его разработкой, отладкой, тестированием, продвижением, подготовкой обучающих курсов, документацией трудятся многочисленные пользователи-энтузиасты из так называемого сообщества R.

В отличие от известных статистических пакетов, например, IBM SPSS Statistics и Statistica, расширяющих свои возможности в лучшем случае один раз в год, а то и в несколько лет, R обновляется практически ежемесячно и даже чаще. На официальном сайте R – всеобъемлющем сетевом архиве CRAN (Comprehensive R Archive Network): <http://cran.r-project.org>, а также на сотнях других сайтов-зеркал, расположенных по всему миру, пользователи всегда могут найти и скачать актуальные версии языка и программной среды R для различных типов операционных систем (Windows, Linux, Mac OS X), ознакомиться с новой интегрированной коллекцией инструментальных средств обработки и документацией по их применению. Правда, в интернет-сообществе документация в основном представлена на английском языке.

Многочисленные алгоритмы эффективных приемов обработки информации, включенные в R, обеспечивают:

ввод данных из различных источников и форматов представления:

обработку, как простых данных, так и больших массивов информации (Big Data) с применением многочисленных встроенных функций, объединенных в более чем 10 тыс. пакетов;

использование в процессе обработки множества инструментальных средств работы с массивами, матрицами, иными сложными конструкциями данных;

наглядность в процессе анализа результатов за счет многочисленных

графических возможностей;

сохранение результатов обработки в виде файлов различных форматов.

Языковые возможности R позволяют пользователям писать собственные программы-скрипты, создавать и использовать расширения-пакеты, представляющие собой не только совокупность функций определенного направления, но и справочную информацию по их применению и реализуемым алгоритмам.

Программа на языке R не имеет четкой, строго установленной структуры. Не требуется оформлять ее стандартное начало и окончание. Обработка программы осуществляется не транслятором, и компоновщиком, а интерпретатором.

В связи с этим процесс исследования и анализа в R стал интерактивным, так как пользователь не только определяет порядок интерпретации операторов, но и может отправлять на интерпретацию не все операторы сразу, а по одному-двум, отслеживая действие каждого из них. Тем самым исчезает необходимость заранее готовить и вводить в программу все исходные параметры задачи.

Ведь всегда, на любом этапе, по мере необходимости или с целью моделирования процесса можно вставить пропущенные значения, выполнить трансформацию данных, добавить или удалить переменные, изменить порядок или способ обработки.

R допускает интегрирование с процедурами, написанными языках C, C++ , Python, FORTRAN и др., использовать возможности R можно, работая с известными статистическими пакетами IBM SPSS Statistics и Statistica.

Как уже было сказано, графические возможности R достаточно разнообразны.

Преимущества и недостатки R

R применяется везде, где необходима работа с данными. Более всего его применяют для статистического анализа – от вычисления средних до временных рядов.

Достоинства языка R заключаются в:

- наличии бесплатной кроссплатформенной среды для простого написания программ;
- богатом арсенале статистических методов. В R реализованы сложные статистические процедуры, еще недоступные в других программах;
- качественной векторной графике, с помощью которой реализованы самые разнообразные и мощные методы визуализации данных из до-

ступных;

- получении данных из разных источников в пригодном для использования виде. R может импортировать данные из самых разных источников, включая текстовые файлы, системы управления базами данных, другие статистические программы и специализированные хранилища данных. R может также записывать данные в форматах всех этих систем;
- существовании более 7000 проверенных прикладных пакетов;
- установке на разные операционные системы, включая Windows, Unix, Linux и Mac OS.

У R есть и немало недостатков. Самый главный из них – это трудность обучения программе. Команд много, вводить их надо вручную, запомнить все трудно, а привычной системы меню нет. Поэтому порой очень трудно найти, как именно сделать какой-нибудь анализ.

Удобным средством вычислений в R является RStudio. RStudio представляет собой бесплатную интегрированную среду разработки (IDE) для R. Благодаря ряду своих особенностей, этот активно развивающийся программный продукт делает работу с R очень удобной.

Второй недостаток R – относительная медлительность. Некоторые функции, особенно использующие циклы, и виды объектов, особенно списки и таблицы данных, работают в десятки раз медленнее, чем их аналоги в коммерческих пакетах. Но этот недостаток преодолевается, хотя и медленно.

Новые версии R умеют делать параллельные вычисления, создаются оптимизированные варианты подпрограмм, работающие много быстрее, память в R используется все эффективнее, а вместо циклов рекомендуется применять векторизованные вычисления.

Структуры и типы данных. Присваивание

R работает с самыми разными структурами данных, включая скаляры, векторы, массивы данных, таблицы данных и списки. Такое большое разнообразие поддерживаемых структур дает языку R большую гибкость в работе с данными.

Типы данных в R бывают числовыми (numeric), целочисленными (integer), текстовыми (character), логическими (logical), комплексными (complex) и необработанными (байты).

Операторами присваивания в R выступают либо =, либо <- (состоящий из двух символов: < и -) – присваивание справа, либо -> – присваивание слева, как поодиночке, так и в последовательности. Присваивание справа

принято среди профессионалов.

Получение помощи по функциям и средствам

R обладает обширными справочными материалами. Встроенная система помощи содержит подробные разъяснения, ссылки на литературу и примеры для каждой функции из установленных пакетов. Справку можно вызвать при помощи функций, перечисленных в табл. 1.

Таблица 1

Функции вызова справки в R

| Функция | Действие |
|--|---|
| help.start() | Общая справка |
| help(<i>–предмет</i>) или ? <i>предмет</i> | Справка по функции <i>предмет</i> (кавычки необязательны) |
| help.search(<i>–предмет</i>) или ?? <i>предмет</i> | Поиск в справке записей, содержащих <i>предмет</i> |
| example(<i>–предмет</i>) | Примеры использования функции <i>предмет</i> (кавычки необязательны) |
| RSiteSearch(<i>–предмет</i>) | Поиск записей, содержащих <i>предмет</i> в онлайн-руководствах и заархивированных рассылках |
| apropos(<i>–предмет</i> , mode= <i>function</i>) | Список всех доступных функций, в названии которых есть <i>предмет</i> |
| data() | Список всех демонстрационных данных, содержащихся в загруженных пакетах |
| vignette() | Список всех доступных руководств по загруженным пакетам |
| vignette(<i>–предмет</i>) | Список руководств по теме <i>предмет</i> |
| library(help="библиотека") | Справка о библиотеке |

Функция help.start() открывает окно браузера с перечнем доступных руководств разного уровня сложности, часто задаваемых вопросов и ссылок на источники. Функция RSiteSearch() осуществляет поиск на заданную тему в онлайн-руководствах и архивах рассылок и представляет результаты в окне браузера. Функция vignette() вызывает список вводных статей в формате PDF. Такие статьи написаны не для всех пакетов.

Пакеты

R в базовой установке уже обладает обширными возможностями. Однако некоторые наиболее впечатляющие опции программы реализованы в дополнительных модулях, которые можно скачать и установить. Существует более 7000 созданных пользователями модулей, называемых пакетами (packages), которые можно скачать с <https://www.r-project.org/>.

Пакеты – это собрания функций R, данных и скомпилированного программного кода в определенном формате. Директория, в которой пакеты

хранятся на компьютере, называется библиотекой.

Все пакеты R относятся к одной из трех категорий: базовые ("base"), рекомендуемые ("recommended") и прочие, установленные пользователем. Получить их список на конкретном компьютере можно, подав команду `library()` или:

```
installed.packages(priority = "base")
installed.packages(priority = "recommended")
packlist<rownames(installed.packages());write.table(packlist,"clipboard",sep="\t", col.names=NA) – полный список пакетов в формате Excel.
```

CRAN Task Views – тематический каталог пакетов.

Команда `search()` выводит на экран названия загруженных и готовых к использованию пакетов.

Для установки пакета используется команда `install.packages()`. Например, пакет MASS содержит набор данных и статистических функций. Этот пакет можно скачать и установить при помощи команды `install.packages("MASS")`. Пакет нужно установить только один раз. Однако, как и любые другие программы, пакеты часто обновляются их разработчиками.

Для обновления всех установленных пакетов используется команда `update.packages()`. Для получения подробной информации об установленных пакетах можно использовать команду `installed.packages()`. Она выводит на экран список всех имеющихся пакетов с номерами их версий, названиями пакетов, от которых они зависят, и другой информацией.

Установить, какие пакеты загружены в каждый момент проводимой сессии, можно, подав команду `sessionInfo()`.

Для использования пакета в текущей сессии программы нужно загрузить его при помощи команды `library()`. Например, для того чтобы использовать пакет MASS, надо ввести команду `library(MASS)`.

Получить список функций каждого пакета можно, например, подав команду `ls(pos = "package:MASS")`.

Получить список аргументов входящих параметров любой функции загруженного пакета можно, подав команду `args()`. Например, `args(lm)`, где `lm()` функция получения линейной регрессионной модели.

R как калькулятор

Первоначально при входе в интерпретатор или IDE RStudio мы видим приглашение к написанию скрипта или диалоговое консольное окно. Самое первое о чём стоит упомянуть, прежде чем писать какой-либо код, это то что

R поддерживает обычные вычисления в виде операций с числами:

Базовые операции:

```
12 + 18
```

```
## [1] 30
```

```
5 * 3
```

```
## [1] 15
```

```
2 ^ 3 # возведение в степень
```

```
## [1] 8
```

```
sqrt(16) # извлечение квадратного корня
```

```
## [1] 4
```

```
16 ^ (1/4) # для корней степени больше 2
```

```
## [1] 2
```

Округление:

```
round(4/3) # округление (до целой части)
```

```
## [1] 1
```

```
round(4/3, 2) # до второго знака после запятой
```

```
## [1] 1.33
```

Математические константы и функции:

```
pi
```

```
## [1] 3.141593
```

```
exp(1) # константа e
```

```
## [1] 2.718282
```

```
exp(2) # e^2
```

```
## [1] 7.389056
```

```
log(exp(1)) # log – натуральный логарифм
```

```
## [1] 1
```

```
log10(100) # log10 – десятичный логарифм
```

```
## [1] 2
```

```
log(4, base = 2) # можем указать основание логарифма (base)
```

```
## [1] 2
```

Все эти команды были написаны и выполнены непременно в консоли в формате диалога с интерпретатором, как на обычном калькуляторе.

Организация данных в R

Основными типами данных в R являются:

- числовой (numeric);

- целочисленный (integer);
- текстовый (character);
- логический (logical) – только два значения: TRUE и FALSE.

Важно: В дробных числах в R в качестве разделителя используется точка.

Создадим переменную x1 и присвоим ей значение 9.5.

```
x1 <- 9.5
is.numeric(x1) # проверим, является ли числом
## [1] TRUE
is.integer(x1) # проверим, является ли целым числом
## [1] FALSE
is.character(x1) # проверим, является ли текстовой переменной
## [1] FALSE
is.logical(x1) # проверим, является ли логической переменной
## [1] FALSE
```

Создадим переменную x2:

```
x2 <- "welcome"
```

Узнаем, какого она типа:

```
class(x2)
## [1] "character"
```

Важно: Если забыли, что делает та или иная функция, можно спросить это у R:

```
?class # так
```

Или так:

```
help(class)
```

Кроме функции class() в R есть функция typeof(), которая тоже возвращает тип данных:

```
typeof(x2)
## [1] "character"
```

Тип переменной можно менять. Например, превратим строку “2” в число 2:

```
two <- "2"
as.numeric(two)
## [1] 2
```

Логические переменные легко превращаются в числовые:

```
u <- TRUE
e <- FALSE
as.numeric(u)
## [1] 1
as.numeric(e)
## [1] 0
```

Конечно, не у любой переменной мы можем поменять тип. Например, строку "abc" превратить в число не получится:

```
as.numeric("abc")
```

Числовые переменные

С числовыми переменными можно делать то же, что и с числами:

```
a <- 25
b <- 15
sum_ab <- a + b # складывать или вычитать
sum_ab
## [1] 40

prod_ab <- a * b # умножать или делить
prod_ab
## [1] 375

power_ab <- a ^ b # возводить в степень и так далее
power_ab
## [1] 9.313226e+20
```

е здесь – это число 10. Запись 9.313226e+20 означает, что число 9.313226 надо умножить на 10²⁰. Такая запись чисел стандартно называется экспоненциальной формой записи числа и используется, в основном для очень больших или очень малых значений.

Если, напротив, R нужно было бы выдать очень маленькое число, 10 стояло бы в отрицательной степени:

```
2/23789
## [1] 8.407247e-05
```

Текстовые переменные (строки)

Что можно делать с текстовыми переменными? Например, в текстовых переменных можно заменять одни символы на другие. Для этого существует

функция sub().

```
group <- "group#1 group#2 group#3"
sub("#","-", group) # (что заменяем, на что заменяем, где заменяем)
## [1] "group-1 group#2 group#3"
```

Однако функция sub() позволяет изменить только первое совпадение. Для того чтобы заменить все встречающиеся в тексте символы, нужно воспользоваться gsub():

```
gsub("#","-", group) # gsub – от global sub
## [1] "group-1 group-2 group-3"
```

Логические выражения

Необходимы для проверки или формулировки условий.

```
x <- 2
y <- 10
```

Привычные выражения:

```
x > y
## [1] FALSE
x < y
## [1] TRUE
x <= y
## [1] TRUE
x == y # для проверки условия равенства – двойной знак =
## [1] FALSE
```

Менее привычные:

```
x != y # отрицание равенства
## [1] TRUE
x & y < 5 # и (одновременно x и y)
## [1] FALSE
x | y < 10 # или (хотя бы один из x и y)
## [1] TRUE
```

Стандартный вывод данных

Результат любого действия в R, записанный в переменную для какой-либо отладочной информации или для проверки правильности выполнения программы можно выводить напрямую в консоль при выполнении программы (скрипта, сценария). Для целей вывода в стандартном R при-

существует два популярных оператора: `print()` и `cat()`.

Функция `print()` проста в использовании и является стандартной функцией для вывода информации в консоль со специальными служебными знаками, помогающими определять, например, используемые структуры данных в выводимой переменной.

Пример использования функции `print()`:

```
x1 <- 15
print(x1)
x2 <- "Голова на плечах"
print(x2)
x3 <- x1 + nchar(x2) # nchar() – количество символов в строке
print(x3)
```

Результат выполнения скрипта:

```
[1] 15
[1] "Голова на плечах"
[1] 31
```

При выводе в консоль информации через `print()` для обычных векторных переменных (в примере `x1`, `x2`, `x3` – векторы размера 1) мы наблюдаем число в квадратных скобках перед выводом переменной. Данное число отображает отладочную информацию о переменных.

Функция `cat()` является аналогом `printf()` в языке C, поскольку выводит информацию как есть без умышленных пробелов:

```
x1 <- 15
cat(x1)
x2 <- "Голова на плечах"
cat(x2)
x3 <- x1 + nchar(x2) # nchar() – количество символов в строке
cat(x3)
```

Результат выполнения скрипта:

```
15Голова на плечах31
```

Для правильного вывода в консоль с помощью функции `cat()` необходимо производить настройку вывода при помощи стандартных приёмов языка C с точностью вывода чисел, числом пробелов, эскейп-последовательностями и т.д.

```
x1 <- 15
cat("\t", x1, "\n")
x2 <- "Голова на плечах"
cat(x2, "\t")
x3 <- x1 + nchar(x2) # nchar() – количество символов в строке
cat(x3, "\n")
```

Результат выполнения скрипта:

```
15
Голова на плечах 31
```

Для склейки переменных различного типа в общий вывод используется стандартная функция `paste()` и её разновидность `paste0()`. На вход данная функция принимает переменные различных типов в неограниченном количестве, перечисляемые через запятую, для того, чтобы склеить всю информацию в строку. Данная функция работает только с векторными переменными.

```
x1 <- 15
x2 <- "Голова на плечах"
x3 <- x1 + nchar(x2) # nchar() – количество символов в строке
print(paste(x1, x2, x3))
```

Результат выполнения скрипта:

```
[1] "15 Голова на плечах 31"
```

Более подробно о работе данной функции будет рассказано далее.

Стандартный ввод данных

В режиме разработки скрипта (сценария) язык R предоставляет интерфейс потокового считывания данных с клавиатуры в консоли на одну или несколько строк, или из потока символов, например при перенаправлении потока символов выполнения программы применяя пайплайн в командной строке `linux`.

Для считывания информации с клавиатуры или из потока символов применяется функция `readLines()`:

```
readLines(con, n, ..., encoding, ...),
```

где `con` – обозначает соединение, из которого считывается поток символов, им по умолчанию без какого-либо указания является ввод символов в консоль, однако может быть указан один из распространённых

текстовых форматов, из которых можно в стандартной или указанной кодировке прочитать поток символов,

`n` – количество строк, которое необходимо считать, если не указано, будет происходить чтение потока вплоть до конца файла (EOF), или до команды завершения ввода в консоли,

`encoding` – стандартная кодировка для считывания символов в переменную.

Результатом присваивания какой-либо переменной выполнения данной стандартной функции сохраняет в ней вектор строк (о векторах подробно будет рассказано позднее), где размер вектора определяется аргументом `n`. Сколько строк, разделённых эскейп-последовательностью “\n”, будет считано из файла, столько элементов и будет в считанной переменной.

Также для данной команды есть и другие аргументы, управляющие работой данной функции, но для стандартной работы с потоком ввода этого достаточно, и все остальные аргументы можно просмотреть при помощи справки.

Файл «connection_text.txt» имеет следующие данные:

Чёрный вечер.

Белый снег.

Ветер, ветер!

На ногах не стоит человек.

Ветер, ветер –

На всем Божьем свете!

Фрагмент скрипта, иллюстрирующего пример считывания первых 4 строк файла:

```
var_poem <- readLines(con = "Practice1/connection_text.txt",  
                      n = 4, encoding = "UTF-8")  
print(var_poem)  
print(nchar(var_poem))
```

Результат выполнения скрипта:

```
[1] " Чёрный вечер."      "  Белый снег."      "   Ветер, ветер!"  
[4] "На ногах не стоит человек."  
[1] 14 15 17 26
```

Результат команды считывания первых четырёх строк из файла состоит из четырёх элементов строкового типа. Результат вывода на экран

количества символов в строке, выводит четыре числа – количество символов для каждой строки в отдельности.

Для считывания из консоли напрямую лишь одной строки применяется другая узкоспециализированная функция `readline(prompt = "")`. Данная функция имеет лишь один аргумент – приглашающую строку, в которой содержится информация для пользователя, какую информацию на вход ожидает программа:

```
> readline(prompt = "Введите число n:")
Введите число n:15
"15"
```

В случае простого считывания строки из консоли, программа просто выведет считанную обратно информацию, как показано в примере выше. Давайте попробуем считать строку из консоли в переменную. Ниже представлен скрипт, выполняющий программу:

```
var_char_name <- readline(prompt = "Введите ваше имя: ")
print(paste("Здравствуйте,", var_char_name))
var_char_age <- readline(prompt = "Сколько вам полных лет?: ")
print(paste0("Вы довольно молодо выглядите для своих ",
             var_char_age,
             ', ',
             var_char_name))
```

Результат выполнения программы с построчным вводом:

```
Введите ваше имя: Иван
[1] "Здравствуйте, Иван"
Сколько вам полных лет?: 22
[1] "Вы довольно молодо выглядите для своих 22, Иван"
```

Рисунок 1 Результат выполнения программы ввода-вывода в консоли

Вывод в текстовый файл

При выполнении программы в формате скрипта или в режиме интерпретатора можно производить вывод содержимого какой-либо переменной или объекта в текстовый файл вместо вывода в консоль. За вывод информации в текстовый файл отвечает функция `write()`:

```
write(x, file = "data",
      ncolumns = if(is.character(x)) 1 else 5,
      append = FALSE, sep = " ")
```

Пример скрипта по записи в файл суммы 4 чисел из другого текстового

файла:

```
?write
xy <- readLines(con = "Practice1/numbers.txt", n = 4) # Считывание чисел
print(xy)      # Отладочная информация о считывании
xy <- as.numeric(xy)    # Преобразование строк в числа
print(xy)
sum_of_numbers <- xy[1] + xy[2] + xy[3] + xy[4]    # Сумма чисел
print(sum_of_numbers)
write(x = sum_of_numbers, file = "Practice1/write_test.txt") # Запись ре-
зультата
```

Результат выполнения скрипта:

```
[1] "56" "98" "4" "56"
[1] 56 98 4 56
[1] 214
```

После этого в указанном пути к файлу записи хранится сумма всех чисел считанных из текстового файла, считанного выше. Обращайте внимание на то, какого типа объявляются объекты переменные при считывании из внешних файлов. Пример выше показывает возможность приведения типа данных после считывания к другому типу для нужд вычислений.

Векторы

Создание векторов

Напомним, вектор — это одномерный массив однотипных данных целых и вещественных чисел, строковых и логических значений. Скалярные величины в памяти R представляются как вектора, состоящие из одного элемента.

Вектор целочисленных значений может быть создан несколькими способами. Варианты создания приведены на рисунке 2.

```

1 # пример 1
2 a<-3
3 b<-6
4 vect1<-a:b # пример 1
5 vect1
6 # пример 2
7 m<--3
8 n<-6
9 vect2<-m:n # пример 1
10 vect2
11 # пример 3
12 vect3<-1:100
13 vect3
14 # пример 4
15 vect4<-c(-8,5,0,45,11)
16 vect4
17 # пример 5
18 vect5<-c(-5:5)
19 vect5

```

Рисунок 2 Варианты создания векторов

Векторы с текстовыми данными могут быть созданы несколькими способами:

```

1 # пример 1
2 vect1<-c("москва", "Екатеринбург", "Сочи", "Ростов")
3 vect1
4 # пример 2
5 vect2<-c(-1:5,10.5,"next")
6 vect2
7 vect2[6]

```

Рисунок 3 Создание векторов с текстовыми данными

Имеются возможности для создания векторов из вещественных чисел:

```

1 # пример 1
2 vect1<-2.5:9.6
3 vect1
4 # пример 2
5 vect2<-seq(5,10,by=0.4)
6 vect2
7 vect2[3] # вывод 3-его элемента вектора
8 f<-1.55
9 vect3<-seq(-2.13,12.5,by=f)
10 vect3

```

Рисунок 4 Создание векторов с вещественными числами

Аналогично можно создать векторы с логическими данными.

Существует еще несколько функций для создания вектора различных типов с инициализированными по умолчанию значениями.

Таблица 2

Функции создания векторов с первоначальными значениями элементов

| Функция | Назначение | Пример использования (фрагмент протокола консоли) |
|--------------|--|--|
| Integer(n) | Создание вектора n целых чисел, все элементы которого равны 0 | > integer(5) [1] 0 0 0 0 0 |
| double(n) | Создание вектора n вещественных чисел, все элементы которого первоначально равны 0 | > double(5) [1] 0 0 0 0 0 > d<-double(5) > d[1]<-4.123456789 > d[1] [1] 4.1234567 |
| numeric(n) | Создание числового вектора n, все элементы которого первоначально равны 0 | > numeric(5) [1] 0 0 0 0 0 |
| complex(n) | Создание вектора n комплексных чисел, все элементы которого первоначально равны 0+0i | complex(5) [1] 0+0i 0+0i 0+0i 0+0i 0+0i s<-complex(5) s[1]<-4.123456769+21i s[1] [1] 4.123457+21i |
| logical(n) | Создание вектора n логических значений, все элементы которого первоначально равны FALSE | ?> logical(4) [1] FALSE FALSE FALSE FALSE |
| character(n) | Создание вектора n строк, все элементы которого первоначально пустые (имеют нулевую длину) | > character(5) [1] "" "" "" "" "" |
| raw(n) | Создание двоичного вектора n элементов, каждый из которых первоначально равен 00 | > raw(5) [1] 00 00 00 00 00 |

Операции над векторами

Над числовыми векторами можно выполнять операции сложения, вычитания, умножения, деления, возведения в степень, изменения знака, сравнения и др. При этом следует иметь в виду, что операции выполняются поэлементно. Операнды могут иметь разное количество элементов: в этом случае для более короткого операнда повторно задействуются элементы с его начала.

Приведем примеры над векторами.

```

1 # пример 1 Сумма векторов
2 vect1<-c(2,3,4,5)
3 vect1
4 vect2<-seq(-3,3)
5 vect2
6 sum<-vect1+vect2
7 sum
8 #пример 2 Разность векторов
9 vect1<-c(2,3,4,5)
10 vect1
11 vect2<-seq(-3,3)
12 vect2
13 razn<-vect1-vect2
14 razn
15 #пример 3 Произведение векторов
16 vect1<-c(2,3,4,5)
17 vect1
18 vect2<-seq(-3,3)
19 vect2
20 pr<-vect1*vect2
21 pr
22 #пример 4 Деление векторов
23 vect1<-c(2,3,4,5)
24 vect1
25 vect2<-seq(-3,3)
26 vect2
27 del<-vect1/vect2
28 del
29 #пример 5 Возведение в степень
30 vect1<-2
31 vect1
32 vect2<-1:10
33 vect2
34 voz<-vect2^vect1
35 voz
36 #пример 6 Изменение знака вектора
37 vect2<-1:10
38 vect1<---vect2
39 vect1

```

Приведем примеры сравнения векторов.

```

3 # Сравнение векторов на операцию не больше
4 vect1<-c(-8,0,5,11)
5 vect1
6 vect2<-c(-3,-2,-1,0,1,2,3,4)
7 vect2
8 sr<-vect2<=vect1
9 sr
10 # Сравнение векторов на операцию не равно
11 vect1<-c(-8,0,5,11)
12 vect1
13 vect2<-c(-3,-2,-1,0,1,2,3,4)
14 vect2
15 sr<-vect2!=vect1
16 sr

```

Рисунок 5 Сравнение векторов разной длины

Для выполнения операции сортировки элементов вектора (по возрастанию или по убыванию) используется функция `sort()`:

`sort(x, decreasing = FALSE,...)`

Первый параметр функции `x` представляет собой вектор, элементы которого должны быть упорядочены,

Второй параметр `decreasing` указывает на сортировку по возрастанию (`FALSE`) или по убыванию (`TRUE`).

Примеры выполнения сортировки вектора:

```

17 # Сортировка вектора по возрастанию
18 x <- c(174, 162, 188, 192, 165, 168, 172)
19 sort_1<-sort(x,decreasing = FALSE)
20 sort_1
21 # Сортировка вектора по убыванию
22 x <- c(174, 162, 188, 192, 165, 168, 172)
23 sort_1<-sort(x,decreasing = TRUE)
24 sort_1

```

Рисунок 6 Выполнение операции сортировки вектора

Над векторами со строковыми данными можно выполнять операции сравнения и принадлежности. Еще раз напоминаем, что операции выполняются поэлементно; в более коротком операнде повторно задействуются элементы вектора с его начала.

```

2 vect1<-c("Антонов А.А.", "Борисов Б.Б.", "Воронин В.В.", "Гусев Г.Г.",
3          "Дмитриев Д.Д.", "Ерохин Е.Е.", "Жук С.А.")
4 vect2<-c("Жук С.А.", "Петров П.П.", "Антонов А.А.", "Романов Р.Р.")
5 vect1
6 vect2
7 # Оценка принадлежности элементов более короткого вектора более длинному
8 rez<-vect2 %in% vect1
9 rez
10 # Оценка принадлежности элементов более длинного вектора более короткому
11 rez<-vect1 %in% vect2
12 rez
13 # Сравнение векторов на неравенство
14 rez2<-vect1!=vect2
15 rez2

```

Рисунок 7 Выполнение операций над строковыми данными

Над строковыми векторами (и не только строковыми) можно выполнять операцию сцепления (конкатенации). Для этого используют функцию `paste()`, которая объединяет несколько векторов после предварительного их преобразования в строковые:

`paste (объект1, объект2,..., объект n, sep = " ", collapse = NULL)`

Параметры объект1, объект2, ..., объект n представляют собой преобразованные в строковые вектора данные, которые планируется сцепить в единый строковый вектор.

Параметр `sep` задает разделитель между сцепляемыми объектами.

Параметр `collapse`, в случае значения, не равного `NULL`, определяет соединитель элементов вектора – результата сцепления в единую строку.

Приведем пример сцепления векторов со строковыми элементами с помощью функции `paste()`.

```

3 vect1<-c("Понедельник", "Вторник", "Среда", "Четверг", "Пятница")
4 vect1
5 vect2<-c("Анализ данных", "Информатика", "Математика", "Статистика",
6          "Теория вероятности")
7 vect2
8 int5<-c(2,4,4,2) # количество часов занятий
9 t<-"час."
10 # 1 вариант вывод единой строки
11 rez<-paste(vect1,vect2,int5,t,sep=" - ",collapse=";")
12 rez
13 rez[2]
14 # вариант поэлементный вывод результата
15 rez<-paste(vect1,vect2,int5,t,sep=" - ")
16 rez

```

Рисунок 8 Выполнение операций над векторами со строковыми данными

Преобразование типов

Как и у обычных переменных, у векторов можно изменять тип (тип всех элементов вектора). Например, можем преобразовать текстовый вектор в числовой с помощью функции `as.numeric()`:

```
text <- c("2", "3", "5")
as.numeric(text)
## [1] 2 3 5
```

При этом если среди элементов есть дробное число, записанное, как текст, то все элементы вектора преобразуются в дробные числа:

```
as.numeric(c("2.3", "6", "8"))
## [1] 2.3 6.0 8.0
```

Замечание: если бы в “2.3” разделителем являлась запятая, ничего бы не получилось – R в качестве разделителя разрядов признает только точку:

```
old <- c("2,3", "6", "8")
as.numeric(old)
## Warning: в результате преобразования созданы NA
## [1] NA 6 8
```

В таком случае нужно сначала заменить запятую на точку с помощью `gsub()`, а уже потом преобразовывать:

```
new <- gsub(",", ".", old) # (что заменяем, на что заменяем, где заменяем)
as.numeric(new)
## [1] 2.3 6.0 8.0
```

Работа с элементами вектора

Для того чтобы выбрать элементы вектора по их индексу (положению в векторе), нужно учитывать, что в R нумерация начинается с 1, а не с 0, как в Python и многих языках программирования.

```
names <- c("Mary", "John", "Peter")
names
## [1] "Mary" "John" "Peter"
names[1] # первый элемент вектора names
## [1] "Mary"
names[0] # не работает
## character(0)
names[1:2] # первые два элемента вектора names
```

```
## [1] "Mary" "John"
```

Если нужно выбрать элементы, которые следуют в векторе не подряд, индексы интересующих нас элементов нужно оформить в виде вектора:

```
names[c(1, 3)] # первый и третий
```

```
## [1] "Mary" "Peter"
```

names[c(1:2, 2:3)] # срезы тоже можно перечислять в качестве элементов вектора

```
## [1] "Mary" "John" "John" "Peter"
```

Обратите внимание: в отличие от Python, в R правый конец среза включается. Например, код names[1:2] вернёт первые два элемента, а не только элемент с индексом 1.

А теперь мы будем отбирать элементы вектора по их значению. Для этого необходимо указывать интересующие критерии выбора (условия) в квадратных скобках. Создадим вектор v:

```
v <- c(1, 8, 9, 2, 3, 0, -1)
```

```
v
```

```
## [1] 1 8 9 2 3 0 -1
```

Выберем элементы вектора v, которые больше 3:

```
v[v > 3]
```

```
## [1] 8 9
```

Усложним задачу. Будем выбирать только четные элементы вектора v. Для этого нам понадобится оператор для определения остатка от деления: %%. Четные элементы – те, которые делятся на 2 без остатка. Значит, остаток от деления их на 2 должен быть равен нулю:

```
v[v%%2 == 0] # только четные элементы
```

```
## [1] 8 2 0
```

Условия можно сочетать:

```
v[v > 3 & v%%2 == 0] # четные элементы больше 3
```

```
## [1] 8
```

Иногда нам нужно не найти элемент вектора по его номеру или по определенным критериям, а выполнить обратную задачу: вернуть индекс элемента (его порядковый номер в векторе). Это можно сделать так:

```
names
```

```
## [1] "Mary" "John" "Peter"
```

```
which(names == 'Jane') # двойной знак =  
## integer(0)
```

Можно также получать индексы элементов вектора, которые удовлетворяют определенным условиям:

```
v  
## [1] 1 8 9 2 3 0 -1  
which(v > 3)  
## [1] 2 3  
which(v%%2 == 0) # индексы четных чисел  
## [1] 2 4 6
```

А как быть, если мы хотим изменить вектор? Например, добавить значение? Все просто:

```
p <- c(1, 2)  
p[3] = 7 # добавим третий элемент  
p  
## [1] 1 2 7
```

Полезный факт: Индекс последнего элемента вектора в R совпадает с длиной вектора.

Если нужно удалить элемент, это делается так:

```
v[v != 8] # хотим убрать 8  
## [1] 1 9 2 3 0 -1
```

Это работает и тогда, когда в векторе встречаются повторяющиеся значения – убираются все совпадающие элементы:

```
w <- c(6, 6, 6, 7)  
w[w != 6]  
## [1] 7
```

Для того чтобы произвести подвыборку без указанных позиций можно использовать «минус» перед перечислением индексов.

```
vector <- c(9, 12, 13, 18, 21, -65, 99)  
print(vector[-c(1, 5)]) # Убираем элементы 1 и 5 позиций  
12 13 18 -65 99
```

Для вывода на экран отсортированного вектора без применения функции принудительной сортировки, т.е. не изменяя характер объекта вектора и порядок хранящихся элементов, можно воспользоваться при индексации

функцией `order()`:

```
vector <- c(9, 12, 13, 18, 21, -65, 99)
print(order(vector))
print(vector[order(vector)])
```

Результат выполнения скрипта:

```
[1] 6 1 2 3 4 5 7
[1] -65  9 12 13 18 21 99
```

Результатом выполнения функции `order()` является возврат вектора индексов элементов вектора, порядок которых определяется возрастанием элементов вектора. При подстановке данного вектора при индексации в исходный вектор мы получаем отсортированный вектор, порядок которого не изменён.

Для функции `order()` также определены некоторые аргументы:

```
order(..., na.last = TRUE, decreasing = FALSE, method = c("auto", "shell",
"radix"))
```

На первом месте находится приём перечислением векторов различного типа с одинаковым размером; `na.last` – позволяет контролировать сортировку пропущенных NA значений, либо впереди, либо в конце вектора; `decreasing` – упорядочивание элементов; `method` – применяемый метод сортировки.

Пример использования функции `order()`:

```
vector <- c(9, 12, 13, 18, 21, -65, 99)
print(order(vector))
print(order(vector, decreasing = TRUE))
print(vector[order(vector, decreasing = TRUE)])
print(vector[order(vector, decreasing = TRUE)[-c(1, 3)])]
```

Результат выполнения скрипта:

```
[1] 6 1 2 3 4 5 7
[1] 7 5 4 3 2 1 6
```

```
[1] 99 21 18 13 12 9 -65
```

```
[1] 21 13 12 9 -65
```

Первая и вторая команды после объявления вектора выводят прямой и обратный порядок сортировки элементов вектора, третья команда выводит элементы вектора по убыванию, четвёртая команда выводит также элементы вектора по убыванию, но без первого и третьего элементов (индексация вектора индексов внутри индексации вектора).

Последовательности

Для создания векторов можно использовать последовательности (для владеющих Python: аналог `range()` и `arange()`, но в отличие от Python, здесь в вектор включаются оба конца). Например, последовательность из целых значений от 0 до 10:

```
0:10
```

```
## [1] 0 1 2 3 4 5 6 7 8 9 10
```

Мы можем сохранить результат в вектор `my_seq`.

```
my_seq <- 1:10
```

```
my_seq
```

```
## [1] 1 2 3 4 5 6 7 8 9 10
```

А вот последовательность из целых значений от 1 до 10 с шагом 2:

```
seq(from = 1, to = 10, by = 2)
```

```
## [1] 1 3 5 7 9
```

Названия параметров можем опускать, если сохраняем их порядок:

```
seq(1, 10, 2)
```

```
## [1] 1 3 5 7 9
```

Шаг также может быть дробным:

```
seq(1, 10, 0.5)
```

```
## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5
```

```
## [15] 8.0 8.5 9.0 9.5 10.0
```

Векторы из повторяющихся значений

В R можно быстро составить вектор из повторяющихся значений. Например, три раза повторить “Repeat me”:

```
rep('Repeat me', 3)
## [1] "Repeat me" "Repeat me" "Repeat me"
```

Или три раза повторить вектор с двумя значениями 0 и 1:

```
rep(c(1, 0), 3)
## [1] 1 0 1 0 1 0
rep(c('Yes', 'No'), each = 4) # повторить 4 раза каждый элемент вектора
## [1] "Yes" "Yes" "Yes" "Yes" "No" "No" "No" "No"
```

Поиск уникальных значений и подсчет значений

А как получить вектор без повторяющихся значений? Для этого есть функция `unique()`:

```
v <- c(1, 1, 0, 2, 2, 3, 5, 1, 8, 2)
unique(v)
## [1] 1 0 2 3 5 8
```

Как посчитать, сколько раз в векторе встречаются различные значения? Для этого есть функция `table()`:

```
table(v)
## v
## 0 1 2 3 5 8
## 1 3 3 1 1 1
```

Функция `table()` возвращает абсолютные частоты значений. Чтобы получить относительные частоты, то есть доли, можно посчитать их самостоятельно:

```
table(v)/sum(table(v))
## v
```

```
## 0 1 2 3 5 8
## 0.1 0.3 0.3 0.1 0.1 0.1
```

И даже перевести в проценты:

```
table(v)/sum(table(v)) * 100
## v
## 0 1 2 3 5 8
## 10 30 30 10 10 10
```

Пропущенные значения

Можно создавать векторы с пропущенными значениями (NAs, от “not applicable”):

```
w <- c(0, 1, NA, NA)
w
## [1] 0 1 NA NA
```

Проверим, являются ли элементы пропущенными значениями:

```
is.na(w) # проверяем, является ли NA
## [1] FALSE FALSE TRUE TRUE
which(is.na(w)) # возвращаем индексы NAs
## [1] 3 4
```

А так не работает, будьте бдительны:

```
which(w == NA)
## integer(0)
```

Обратите внимание: NA указывается без кавычек! Это не текст, который кодирует пропущенные значения, а особый «тип» данных (наличие NA не изменяет тип переменной, то есть, если NA встречаются в числовой переменной, переменная будет восприниматься R как числовая).

Самостоятельная работа №1

Часть 1

Задание 1.

В двух переменных сохранены некоторые значения:

```
x <- 2  
y <- 4
```

Напишите код, который позволит поменять значения в переменных x и y местами, то есть получить следующее:

```
x  
## [1] 4  
y  
## [1] 2
```

Внимание: Ваш код должен работать для любых значений x, y.

Задание 2.

```
x <- 3.5  
y <- "2,6"  
z <- 1.78  
h <- TRUE
```

- Определите типы переменных.
- Сделайте переменную h целочисленной.
- Сделайте переменную y числовой (обратите внимание на запятую!).
- Сделайте переменную x текстовой.

Задание 3.

Исследователь сохранил доход респондента в переменную dohod:

```
dohod <- 1573
```

Исследователь передумал и решил изменить значение этой переменной – сохранить в нее натуральный логарифм дохода. Помогите ему!

Задание 4.

Создать текстовый файл в папке проекта. В текстовом файле указать номер варианта по списку группы. Считать данные из текстового файла. Вывести в консоль число $2*N - 1$, где N – считанный номер варианта.

Часть 2

Задание 1.

Дан вектор g , в котором хранятся следующие значения:

1, 0, 2, 3, 6, 8, 12, 15, 0, NA, NA, 9, 4, 16, 2, 0

Выведите на экран:

- первый элемент вектора
- последний элемент вектора
- элементы вектора с третьего по пятый включительно
- элементы вектора, которые равны 2
- элементы вектора, которые больше 4
- элементы вектора, которые кратны 3 (делятся на 3 без остатка)
- элементы вектора, которые больше 4 и кратны 3
- элементы вектора, которые или меньше 1, или больше 5
- индексы элементов, которые равны 0
- индексы элементов, которые не меньше 2 и не больше 8
- элементы вектора по возрастанию, с пропущенными значениями в конце без цифр «2»

Задание 2.

Напишите код, который заменяет последний элемент вектора на пропущенное значение (NA). Ваш код должен работать для любого вектора (любой длины).

Задание 3.

Напишите код, который выводит на экран индексы пропущенных значений в векторе.

Задание 4.

Напишите код, который считает, сколько пропущенных значений в векторе.

Задание 5.

Напишите код, который позволяет создать вектор из id (уникальных номеров) респондентов, если известно, что в опросе участвовало 100 респондентов.

Задание 6.

Известно, что в базе данных хранятся показатели по 3 странам за 5 лет. Таблица имеет вид:

Таблица 3

Пример таблицы

| № | country | year |
|-----|---------|------|
| 1. | France | 2019 |
| 2. | France | 2020 |
| 3. | France | 2020 |
| 4. | France | 2018 |
| 5. | France | 2017 |
| 6. | Italy | 2019 |
| 7. | Italy | 2020 |
| 8. | Italy | 2020 |
| 9. | Italy | 2018 |
| 10. | Italy | 2017 |
| 11. | Spain | 2019 |
| 12. | Spain | 2020 |
| 13. | Spain | 2020 |
| 14. | Spain | 2018 |
| 15. | Spain | 2017 |

- Создайте вектор с названиями стран (первый столбец).
- Создайте вектор, который мог бы послужить вторым столбцом в таблице, представленной выше (подумайте, какую длину имеет этот вектор).

Задание 7.

Исследователю из задачи 3 из части 1 понравилось, как Вы работаете в R, и теперь он решил создать вектор `income`, в котором сохранены доходы нескольких респондентов:

```
income <- c(10000, 32000, 28000, 150000, 65000, 1573)
```

Исследователю нужно получить вектор `income_class`, состоящий из 0 и 1: 0 ставится, если доход респондента ниже среднего дохода, а 1 – если больше или равен среднему доходу.

Подсказка: сначала можно посчитать среднее значение по вектору `income` и сохранить его в какую-нибудь переменную. Пользоваться встроенной функцией `mean()` нельзя.

Задание 8.

Создать текстовый файл в папке проекта с именем “coords.txt”. Задать в текстовом файле числами координаты N-мерного вектора \vec{x} , где каждая координата будет расположена в своей строке:

3

15

...

Считать данные из текстового файла. Подсчитать L^P -норму по формуле:

$$\|\vec{x}\|_P = \sqrt[P]{\sum_{i=1}^N |x_i|^P}$$

Размер вектора N и порядок нормы P указан в варианте заданий в таблице:

| № Вар | N | P | № Вар | N | P | № Вар | N | P |
|-------|----|------|-------|----|------|-------|----|------|
| 1 | 5 | 6.21 | 11 | 16 | 1.26 | 21 | 11 | 1.69 |
| 2 | 7 | 4.49 | 12 | 5 | 2.87 | 22 | 19 | 4.09 |
| 3 | 6 | 2.87 | 13 | 4 | 4.68 | 23 | 13 | 3.70 |
| 4 | 8 | 5.50 | 14 | 4 | 4.49 | 24 | 9 | 4.13 |
| 5 | 10 | 2.32 | 15 | 7 | 4.43 | 25 | 14 | 4.40 |
| 6 | 4 | 3.15 | 16 | 14 | 5.76 | 26 | 5 | 2.52 |
| 7 | 6 | 1.34 | 17 | 17 | 1.74 | 27 | 7 | 1.88 |
| 8 | 9 | 4.02 | 18 | 18 | 2.75 | 28 | 19 | 5.57 |
| 9 | 16 | 3.36 | 19 | 12 | 5.53 | 29 | 6 | 2.26 |
| 10 | 9 | 4.29 | 20 | 15 | 5.75 | 30 | 18 | 6.28 |

Вывести результат вычисления нормы в файл “result.txt”.

Задание 9.

Создать текстовый файл в папке проекта с именем “coords.txt”. Задать в текстовом файле числами координаты N-мерного вектора \vec{x} , где каждая координата будет расположена в своей строке:

3

15

...

Считать данные из текстового файла. Подсчитать первые и вторые разности вектора \vec{x} :

$$\Delta x_i = x_{i+1} - x_i$$
$$\Delta^2 x_i = \Delta x_{i+1} - \Delta x_i$$

Подумайте, что должна возвращать программа при $N \leq 2$.

Размер N-мерного вектора остаётся распределённым по каждому варианту по таблице из задания 8.

Результат вычисления векторов записать в файл “diff_vectors.txt”.

Контрольные вопросы:

1. Привести примеры операций над векторами
2. Привести примеры функций создания и обработки векторов

Практическая работа № 2

Матрицы

Создание массивов и матриц данных

Понятия матрицы и массива данным в языке R, как и в других языках программирования, а также операции над ними не идентичны данным понятиям и действиям в математике.

В R матрица – это двумерный массив однотипных данных – чисел, строк или логических значений. Массив данных – это структура однотипных данных с размерностью больше двух.

Подходы по созданию в R матриц и массивов данных на основе существующих векторов идентичны. Матрицы в R можно создавать разными способами. Выбор способа зависит от того, какую матрицу мы хотим создать: пустую матрицу (чтобы потом заполнять ее нужными значениями) или матрицу, составленную из уже имеющихся значений, например, из векторов.

```
2 # Создание матриц и массивов
3 z<-1:30
4 dim(z)<-c(3,10)# преобразование вектора в 2-мерную матрицу
5 z
6 z[2,10]# вывод 10-го элемента 2-ой строки
7
8 z1<-1:60
9 dim(z1)<-c(3,10,2)# преобразование вектора в 3-мерный массив
10 z1
11 z1[2,10,1]#вывод элемента массива данных
```

Рисунок 9 Пример создания матриц и массивов

Для того чтобы создать пустую матрицу, нужно использовать функцию `matrix()`. Размерность матрицы – число строк и число столбцов в ней.

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE,
       dimnames = NULL)
```

Параметр `data` может содержать значения создаваемой матрицы, заданные любым способом. По умолчанию (при значении `NA`) создается пустая матрица. Параметры `nrow` и `ncol` задают количество строк и столбцов в создаваемой матрице (соответственно).

Параметр `byrow` определяет порядок заполнения матрицы, при значении `FALSE` (по умолчанию) матрица заполняется по столбцам (сверху вниз), в противном случае – по строкам.

Параметр `dimnames` представляет собой Список, первый элемент которого задает имена строк, а второй – имена столбцов создаваемой матрицы.

При значении NULL имена строк и столбцов нумеруются цифрами. Аналогично, если список не полный, одно из измерений нумеруется цифрами.

Создадим матрицу 2×3 , состоящую из нулей:

```
M <- matrix(0, nrow = 2, ncol = 3)
```

```
M
```

```
##      [,1] [,2] [,3]
```

```
## [1,]  0   0   0
```

```
## [2,]  0   0   0
```

Можем посмотреть на ее размерность:

```
dim(M)
```

```
## [1] 2 3
```

Рассмотрим, как собрать матрицу из «готовых» векторов. Пусть у нас есть три вектора

```
x <- c(1, 2, 3, 0)
```

```
y <- c(4, 5, 6, 0)
```

```
z <- c(7, 8, 9, 0)
```

и мы хотим объединить их в матрицу. Векторы будут столбцами матрицы:

```
m_cols <- cbind(x, y, z) # c – от columns
```

```
m_cols
```

```
##      x y z
```

```
## [1,] 1 4 7
```

```
## [2,] 2 5 8
```

```
## [3,] 3 6 9
```

```
## [4,] 0 0 0
```

А теперь векторы будут строками матрицы:

```
M_rows <- rbind(x, y, z) # r – от rows
```

```
M_rows
```

```
##      [,1] [,2] [,3] [,4]
```

```
## x      1   2   3   0
```

```
## y      4   5   6   0
```

```
## z      7   8   9   0
```

```
2 # Создание заполненной матрицы 3*2 с именованием строк
3 m1<-matrix(data=1:6,nrow = 3,ncol = 2,byrow = FALSE,
4           dimnames = list(c("str1", "str2","str3")))
5 m1
6 # Создание заполненной матрицы 4*2 с именованием строк и столбцов
7 m2<-matrix(data=1:8,nrow = 4,ncol = 2,byrow = TRUE,
8           dimnames = list(c("str1", "str2","str3","str4"),c("col1","col2")))
9 m2
```

Рисунок 10 Создание матрицы с помощью функции matrix()

Еще одна функция – `array()` позволяет создавать как матрицы, так и массивы данных любой размерности:

```
array (data = NA, dim = length(data), dimnames = NULL)
```

Параметр `dim` определяет размерность создаваемого объекта.

Значение параметра `dimnames` аналогично, описанному для функции `matrix()`

Примеры использования функции `array()` для создания многомерных объектов с элементами одного и того же типа приведены на рисунке .

```
11 # Создание заполненной матрицы 2*3:
12 m3<-array(1:12,c(2,3))
13 m3
14 # Создание заполненного массива данных 3*4*2:
15 m4<-array(1:48,c(3,8,2))
16 m4
17 # Создание поименованного массива данных 3*4*2:
18 m5<-array(data=1:48,dim = c(3,6,2),dimnames = list(c("str1", "str2","str3"),
19                                                     c("r1","r2","r3","r4","r5","r6"), c("z1","z2")))
20 m5
21 # Создание пустого массива данных 3*4*2:
22 m6<-array(dim=c(3,6,2))
23 m6
```

Рисунок 11 Создание объектов с помощью функции `array()`

Элементы матрицы

Для того чтобы обратиться к элементу матрицы, необходимо указать строку и столбец, на пересечении которых он находится:

```
m1[1, 3]
```

```
## [1] 7
```

Если нам нужна отдельная строка (одна строка, все столбцы), то номер столбца нужно не указывать, просто оставить позицию пустой:

```
m1[1, ] # вся первая строка
```

```
## A B C D
```

```
## 1 4 7 0
```

Аналогично для столбцов:

```
m1[, 2] # весь второй столбец
```

```
## r1 r2 r3
```

```
## 4 5 6
```

Операции над матрицами

Над матрицами и массивами данных с числовыми значениями можно выполнять те же операции, что и для векторов, причем с теми же уточнениями. Однако если планируется решение математических задач, следует

соблюдать требования, установленные в математике, в частности, по размерностям объектов и значениям их элементов.

Транспонирование матрицы

Транспонированной называется матрица $A_{(m \times n)}^T$, полученная из исходной матрицы $A_{(m \times n)}$ путем замены строк на столбцы, а столбцов на строки.

Для вычисления транспонированной матрицы используется функция `t()`, единственным аргументом которой является исходная матрица $A_{(m \times n)}$.

Пример использования функции `t()` приведен на рисунке 11.

```
2 # Транспонирование матрицы с помощью функции t()
3 a<-matrix(data=1:12,nrow = 3,ncol = 4,byrow = FALSE)
4 a
5 # Вычисление транспонированной матрицы
6 at<-t(a)
7 at
```

Рисунок 12 пример выполнения транспонирования матрицы

Обратная матрица

Обратной называется матрица $A_{(m \times n)}^{-1}$, при умножении на которую исходная квадратная матрица $A_{(n \times n)}$ дает в результате единичную матрицу $E_{(n \times n)}$. Обратная матрица существует тогда и только тогда, когда определитель исходной матрицы не равен нулю

Для вычисления обратной матрицы применяют функцию `solve()` с единственным аргументом – исходной матрицей $A_{(n \times n)}$.

Пример вычисления обратной матрицы приведен на рисунке 12.

```
9 # Вычисление обратной матрицы с помощью функции solve()
10 a1<-matrix(data=c(1,6,2,7),nrow = 2,ncol = 2,byrow = FALSE)
11 a1
12 ao<-solve(a1)
13 ao
```

Рисунок 13 Пример вычисления обратной матрицы

Перемножение матриц

Для перемножения матриц по правилам математики используется операция `%*%`. Примеры перемножения матриц приведены на рисунке 13.

```

15 # Перемножение матриц с помощью операции %*%
16 a1<-matrix(data=c(1,6,2,7),nrow = 2,ncol = 2,byrow = FALSE)
17 a1
18 ao<-solve(a1)
19 ao
20 malt<-a1 %*% ao
21 malt
22 # Умножение матрицы на транспонированную матрицу
23 a<-matrix(data=1:12,nrow = 3,ncol = 4,byrow = FALSE)
24 a
25 # Вычисление транспонированной матрицы
26 at<-t(a)
27 at
28 rez<-a %*% at
29 rez

```

Рисунок 14 Пример выполнения операции перемножения матриц

Диагональная матрица

Диагональной называется квадратная матрица, все элементы которой, стоящие вне главной диагонали, равны нулю.

Для создания диагональной матрицы используется функция `diag()`.

`diag(x = 1, nrow, ncol, names = TRUE)`

Параметр `x` задает значения диагональных элементов (по умолчанию создается единичная диагональная матрица).

Параметры `nrow` и `ncol` определяют число строк и столбцов; значения данных параметров, как правило, должны быть равны, хотя это не обязательно.

Параметр `names` при значении `TRUE` предписывает, что в создаваемой диагональной матрице должны наследоваться имена исходной матрицы.

Пример создания диагональной матрицы приведен на рисунке 14.

```

2 # Создание диагональной матрицы
3 ? diag
4 # 1 вариант
5 nrow<-5
6 ncol<-5
7 e_1<-diag(x=1,nrow,ncol,names = T)
8 e_1
9 # 2 вариант
10 # вывод диагональной матрицы
11 y=diag(1:4)
12 y
13 # 3 вариант
14 # количество строк не равно числу столбцов
15 z<-diag(nrow=4,ncol=5)
16 z

```

Рисунок 15 Создание диагональных матриц

Собственные векторы и собственные значения

Собственный вектор квадратной матрицы $A_{(n \times n)}$ представляет собой вектор X_n , умножение матрицы $A_{(n \times n)}$ на который дает коллинеарный вектор λX_n :

$$A_{(n \times n)} X_n = \lambda X_n,$$

где λ – скалярная величина собственное значение вектора X_n относительно матрицы $A_{(n \times n)}$.

Для вычисления собственных значений и собственных векторов числовых (двойных, целочисленных, логических) или комплексных матриц используется функция `eigen()`;

```
eigen(x, symmetric, only.values = FALSE, EISPACK = FALSE)
```

Раскроем назначение основных параметров функции.

Параметр `x` представляет собой матрицу с целочисленными, вещественными или комплексными значениями элементов, спектральное разложение которой должно быть вычислено. Логические матрицы приводятся к числовым (`TRUE = 1`, `FALSE = 0`).

Если параметр `only.values` равен `TRUE`, вычисляются и возвращаются только собственные значения, иначе возвращаются как собственные значения, так и собственные вектора.

```
2 # вычисление собственных значений и собственных векторов
3 y<-diag(c(2,4,1,6))
4 y
5 e<-eigen(y)
6 e$values # вывод собственных значений
7 e$vectors # вывод собственных векторов
```

Рисунок 16 Вычисление собственных значений и собственных векторов матрицы

Приведем примеры обращений к элементам матриц и массивов данных.

По ходу изложения материала варианты обращений к элементам матриц и массивов данных приводились неоднократно. Еще некоторые возможности представлены на рисунке 17.

```
9 # обращение к элементам матриц и массивов данных
10 w<-array(1:24,dim = c(3,3,2))
11 w
12 w[1,,2]
13 m<-matrix(data = c(1:4,5,6,12,9,4),nrow = 3,ncol = 3)
14 m
15 m[1,]
16 m[,1]
17 m[1:2,1:2]
```

Рисунок 17 Примеры обращения к элементам матриц и массивов

Метаданные матриц

В R любая переменная является объектом какого-либо класса, например стандартные числа – это векторы длины 1, матрицы – это векторы векторов и т.д. Для матриц, как и для векторов, определены специальные функции, позволяющие работать со свойствами матриц как со свойствами объектов класса.

Имена строк и столбцов. Допустим, вы создали матрицу по специальной формуле или заданию и вам необходимо обозначить в данной матрице названия строк и столбцов. Для решения данной задачи используются функции просмотра свойств матрицы `rownames()` и `colnames()`.

Данные функции при вызове их относительно какой-то заданной матрицы будут возвращать их действительные имена строк и столбцов, а также если при вызове функции, присваивать данному вызову вектора строк той же длины, что и данная размерность, то они присваиваются на соответствующие места в качестве имён строк или столбцов. Пример использования данных функций:

```
# инициализируем матрицу
matrix <- diag(1:4)
print(matrix)
```

```
# попробуем посмотреть на имена строк и столбцов
cat("\nСтроки:\n")
print(rownames(matrix))
cat("\nСтолбцы:\n")
print(colnames(matrix))
```

```
# присвоим имена
cat("\nПрисвоение названий\n")
rownames(matrix) <- paste0("row", 1:4)
colnames(matrix) <- paste0("col", 1:4)
cat("Присвоение завершено\n")
```

```
# попробуем посмотреть на имена строк и столбцов
cat("\nСтроки:\n")
print(rownames(matrix))
```



```
cat("\nСтолбцы:\n")
print(colnames(matrix))
```

Результат выполнения скрипта:

```
  [,1] [,2] [,3] [,4]
[1,]  1  0  0  0
[2,]  0  2  0  0
[3,]  0  0  3  0
[4,]  0  0  0  4
```

Строки:

NULL

Столбцы:

NULL

Присвоение названий

Присвоение завершено

Строки:

```
[1] "row1" "row2" "row3" "row4"
```

Столбцы:

```
[1] "col1" "col2" "col3" "col4"
```

В данном примере показано, как применение свойств переменных в разрезе изменения их метаданных или полей объектов, так и продемонстрирована векторизованная функция `paste()` и её применение в разрезе уменьшения размера кода.

Вычисления по матрице. Для специальных быстрых вычислений по матрице в R существуют функции «обхода» – векторизованные функции быстро вычисляющие агрегацию по столбцам или строкам. Самым часто используемым в таких вычислениях являются функции `rowSums()` и `colSums()`:

```
> # Задание матрицы
> matrix <- cbind(1:4, 9:6, c(1.8, 9.1, -2.3, 3.4))

> print(matrix)
```

```
      [,1] [,2] [,3]  
[1,]   1   9 1.8  
[2,]   2   8 9.1  
[3,]   3   7 -2.3  
[4,]   4   6 3.4
```

```
> # Агрегации
```

```
> print(rowSums(matrix)) # Сумма элементов в строках  
[1] 11.8 19.1 7.7 13.4
```

```
> print(colSums(matrix)) # Сумма элементов в столбцах  
[1] 10 30 12
```

Помимо классических агрегаций по размерностям матрицы кроме суммы можно задать собственную агрегацию путём использования функции `apply()` – функция быстрого вычисления функции по матрице. Функция состоит в применении указанной функции, или созданной на ходу, к указанной размерности матрицы:

```
> print(apply(X = matrix, MARGIN = 1, FUN = sum))  
[1] 11.8 19.1 7.7 13.4
```

```
> print(apply(matrix, 2, sum))  
[1] 10 30 12
```

```
> # Другие агрегации  
> print(apply(matrix, 1, prod))  
[1] 16.2 145.6 -48.3 81.6
```

```
> print(apply(matrix, 2, mean))  
[1] 2.5 7.5 3.0
```

```
> print(apply(matrix, 2, max))  
[1] 4.0 9.0 9.1
```

Из примеров наглядно видно, что `MARGIN` отвечает за размерность к которой применяется функция, в `FUN` указывается имя применяемой

функции или сама функция, X – это собственно матрица.

Самостоятельная работа №2

Часть 1

1. Создайте матрицу размерности 3 * 4, состоящую из 3, а затем измените некоторые ее элементы так, чтобы получить следующее:

2. [3 3 4 3]

3. [1 3 3 3]

[3 NA 3 1]

4. Создайте из следующих векторов матрицу, такую, что:

- векторы являются столбцами матрицы
- векторы являются строками матрицы

```
a <- c(1, 3, 4, 9, NA)
```

```
b <- c(5, 6, 7, 0, 2)
```

```
c <- c(9, 10, 13, 1, 20)
```

Дайте (новые) названия строкам и столбцам матрицы.

3. Может ли матрица состоять из элементов разных типов?

Проверьте: составьте матрицу из следующих векторов (по столбцам):

```
names <- c("Jane", "Michael", "Mary", "George")
```

```
ages <- c(8, 6, 28, 45)
```

```
gender <- c(0, 1, 0, 1)
```

Если получилось не то, что хотелось, подумайте, как это можно исправить, не теряя информации, которая сохранена в векторах.

Добавьте в матрицу столбец `age_sq` – возраст в квадрате.

4. Создайте из векторов из задачи 3 список (list) и назовите его `info`.

- Выведите на экран имя `Michael` (обращаясь к элементам списка, конечно).

- Выведите на экран вектор `gender`.

- Назовите векторы в списке `name`, `age`, `gender`. Выведите на экран элементы вектора `name`.

- Добавьте в список вектор `drinks`, в котором сохранены значения: `juice`, `tea`, `rum`, `coffee`.

- Добавьте в список данные по еще одному человеку: `John`, 2 года, мужской пол, любит молоко.

5. В R есть функция `strsplit()`, которая позволяет разбивать строки (текстовые переменные) на части по определенным символам.

Пусть у нас есть строка s:

```
s <- "a,b,c,d"
```

Мы хотим получить из нее вектор из 6 букв. Применяю функцию:

```
let <- strsplit(s, ",")
```

Получили почти то, что хотели. Почему почти? Потому что получили не вектор, а список!

```
class(let)
```

```
## [1] "list"
```

Превратим в вектор:

```
unlist(let)
```

```
## [1] "a" "b" "c" "d"
```

Теперь все в порядке, получили вектор из четырех элементов.

Дана строка index:

```
index <- "0,72;0,38;0,99;0,81;0,15;0,22;0,16;0,4;0,24"
```

Получите из этой строки числовой вектор I.

Часть 2

Итерационный метод

1. Создать диагональную матрицу A размерности 2x2, состоящую из элементов 4 и 9. Задать для данной матрицы названия строк как «eq1» и «eq2», а для столбцов «x1» и «x2».

2. Найти собственные значения матрицы A, вывести эти значения на экран. Объяснить, почему собственные значения матрицы A получились именно такими.

3. Найти матрицу B с помощью соотношения $B = I - A$, где I — единичная матрица. Вывести данную матрицу на экран.

4. Задать векторы u и f, равными:

$$f = \begin{pmatrix} 4 \\ 2 \end{pmatrix}, u = \begin{pmatrix} 0.2 \\ -0.3 \end{pmatrix}$$

5. Для заданных A, u, f решить СЛАУ вида:

$$A * u = f$$

при помощи стандартного вида решения СЛАУ для невырожденной матрицы A с помощью стандартных функций R. Вывести итоговый вектор решения СЛАУ u_result на экран (вам может понадобиться функция solve()).

6. Произвести 7 итераций по следующей схеме:

$$u_{i+1} = Bu_i + f,$$

где результат каждой итерации будет записан в отдельную переменную и храниться в памяти сессии.

7. Сравните результаты u7 и u_result. Насколько векторы различны по

каждой из координат?

8. Для матрицы A и вектора f произвести деление всех их элементов на максимальное значение элементов матрицы A .

9. Повторить пункты 2-3 и далее 5-7 для полученных в результате пункта 8 матрицы и вектора свободных членов.

10. Сравнить результаты, полученные в пункте 7 и в пункте 9.

Часть 3

Срезы матрицы

Введем матрицу с помощью следующего скрипта:

```
step <- 1          # Шаг сетки
dekart_begin <- -5  # Начало сетки
dekart_end <- 5     # Конец сетки

# Задание сеточной поверхности
x <- seq(from = dekart_begin, to = dekart_end, by = step)
y <- x

# Задание двумерной функции на координатной сетке
surface_matrix <- outer(X = x,
                        Y = y,
                        FUN = function(x,y) Re(exp(-1i * 0.5 * x * y)))
dimnames(surface_matrix) <- list(x, y)
```

Задание 1.

1. Вывести в файл “summary.txt” следующую информацию о созданной матрице с соответствующими подписями к ней:

- количество элементов матрицы (“number of matrix elements:”)
- размерность строк (“number of rows:”)
- размерность столбцов (“number of cols:”)
- сумма элементов главной диагонали (“sum of main diag elements:”)
- сумма элементов срединного среза матрицы по строкам (“sum of middle row elements: ”)
- сумма элементов срединного среза матрицы по столбцам (“sum of middle column elements:”)
- суммы строк матрицы (“row sums:”)
- суммы столбцов матрицы (“col sums:”)

2. Переписать скрипт пункта 1 так, чтобы данные о начале, конце и шаге координатной сетки вводились пользователем с помощью консольного ввода (берём в рассмотрение только квадратную сетку). Произвести вывод в файл “summary2.txt” той же информации о полученной матрице, как и в пункте 1, за исключением суммы элементов срединного столбца и строки (из-за неопределённости в четности или нечетности измерений матрицы)

3. Переписать скрипт пункта 2 так, чтобы данные о начале, конце и шаге координатной сетки вводились пользователем с помощью текстового файла “inputs.txt”. Выбор формата считывания данных остаётся за программистом. В данном задании необходимо переписать скрипт так, чтобы была возможность задавать неквадратную и неравномерную сетку для функции (6 параметров вместо 3). Произвести вывод в файл “summary3.txt” той же информации о полученной матрице, как и в пункте 2.

Часть 4

Машины

Для данного задания в матрицу собраны данные cars о максимальной скорости и тормозной дистанции машин 1920-х годов. На основе данной матрицы данных нам необходимо будет на практике научиться приложениям матричных вычислений в программировании на R.

Если производить специальную визуализацию предоставленных нам данных, то график разброса показаний дистанции торможения от максимальной скорости авто в милях в час выглядит следующим образом (рис.18):

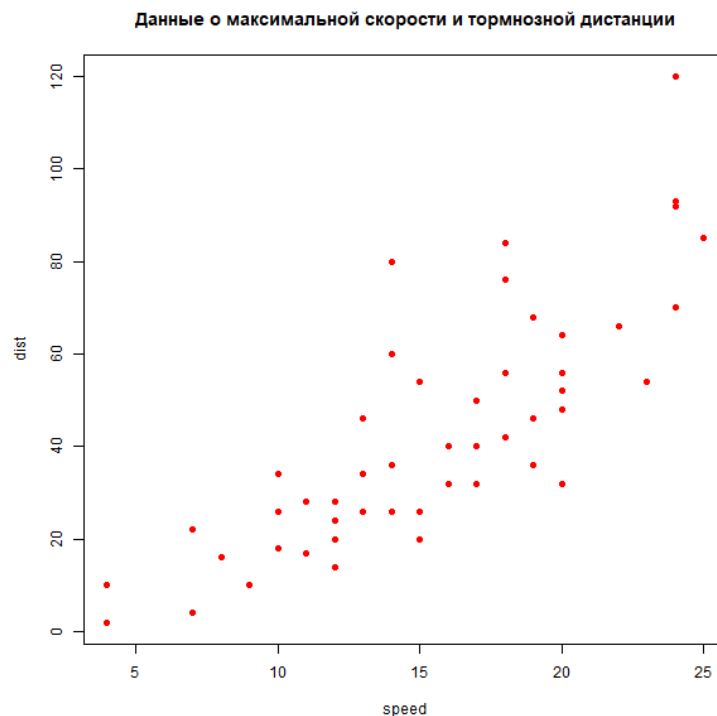


Рисунок 18. Максимальная скорость (x), тормозная дистанция (y) для машин 1920-х годов

Матричные вычисления в данной работе помогут нам построить модель линейной аппроксимации данных при помощи метода наименьших квадратов и нормального уравнения.

Данные для задания формируются в матрицу с помощью следующей команды:

```
cars_matrix <- as.matrix(cars)
```

Задание 1.

1. Создать новую матрицу `cars_speed`, состоящую из 2 столбцов: единичного вектора и первого столбца матрицы `cars_matrix`.

2. Создать новый вектор `cars_dist`, который является срезом матрицы `cars_matrix` по второму столбцу.

3. Рассчитать новый вектор `alpha` по следующему соотношению, называемому *нормальным уравнением модели регрессии*:

$$\vec{\alpha} = (A^T * A)^{-1} * A^T * y,$$

где A – матрица `cars_speed`, y – вектор `cars_dist`, умножение между членами подразумевается под матричным.

4. Проверить тип данных вычисленного вектора `alpha`, произвести преобразование данной переменной к структуре данных “vector”, в случае если получилось иное, без потери, содержащейся в нём информации.

5. Создать переменные `alpha_c` и `alpha_x` на основе первого и второго элемента вектора `alpha` соответственно. Вывести на экран данные вектора по следующему шаблону:

“`alpha_c = ...`” – первый элемент вектора

“`alpha_x = ...`” – второй элемент вектора

6. Создать новый вектор `cars_speed_lm` на основе матрицы `cars_matrix` с помощью взятия его первого столбца.

7. Создать новый вектор `cars_dist_lm` на основе следующего соотношения:

$$\text{cars_dist_lm} = \text{alpha_c} + \text{cars_speed_lm} * \text{alpha_x}$$

8. В вектор `dist_residuals` сохранить информацию о разности между векторами `cars_dist_lm` и `cars_dist`.

9. Вычислить среднее и стандартное отклонение вектора `dist_residuals`.

10. Вывести на экран значения вектора `cars_dist_lm`, убедиться в их отсортированности по возрастанию (потому что это прямая линия).

11. Вывести значения среднего и стандартного отклонений `dist_residuals` на экран.

Результат работы с матрицами:

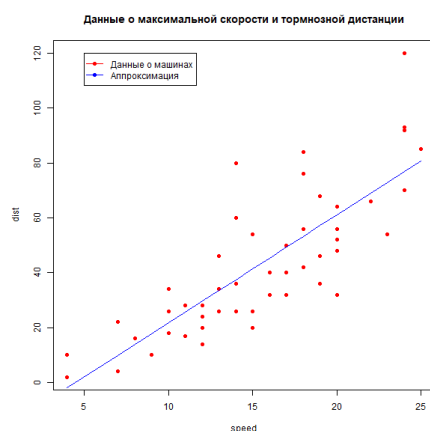


Рисунок 19. Аппроксимирующая прямая для зависимости дистанции торможения от тормозного пути автомобиля

Контрольные вопросы

1. Приведите способы создания матриц
2. Чем массивы данных отличаются от матриц
3. Приведите примеры операций над матрицами и массивами
4. Сравните возможности выполнения операций над матрицами и массивами в математике и в R

Практическая работа №3

Создание и обработка фреймов

Фрейм (или таблица данных) представляет собой двухмерную структуру, в которой строки имеют одинаковую длину, а значения разных столбцов могут принадлежать различным типам – быть числами допустимых форматов, тестовыми или логическими величинами.

Одним из способов создания фрейма является присвоение источника, в качестве которого может быть текстовый документ, файлы табличных процессоров, баз данных, других специальных программ, либо функции, используемые в таких случаях, – `read.table()`, `read.xlsx()`, `read.csv()`, а также приведены примеры их использования.

Другим способом создания фреймов является применение функции `data.frame()`, обеспечивающей формирование таблицы данных из указанных векторов и матриц, записываемых в требуемом порядке следования: `data.frame(x1,x2,...,xn, row.names = NULL, check.rows = FALSE, check.names = TRUE, fix.empty.names = TRUE, stringsAsFactors = default.stringsAsFactors())`

Рассмотрим назначение основных параметров функции. Параметры `x1`, `x2`, `xn` указывают на имена соответствующих векторов и матриц создаваемого фрейма. Данные параметры могут записываться как в форме позиционных, так и ключевых параметров (в виде `X1=x1`, `X2=x2`, `Xn=xn`, где `X1`, `X2`, ..., `Xn` – имена столбцов в создаваемом фрейме).

Параметр `row.names` представляет собой строковый или целочисленный вектор, определяющий имена строк в создаваемом фрейме. По умолчанию (при значении `NULL`) строки нумеруются числами.

Параметр `check.rows` при значении `TRUE` определяет необходимость проверки строки создаваемого фрейма на согласованность длины и имен.

Параметр `check.names` при значении `TRUE` предписывает проверять имена переменных (столбцов) во фрейме на синтаксическую допустимость и отсутствие дублирования. При необходимости имена корректируются.

Для обращения к элементам фрейма следует указывать либо имя фрейма, за которым в квадратных скобках приводить индексы элемента фрейма как плоской структуры данных, либо записывать составное имя, связывая имя фрейма и имя столбца (вектора, матрицы) знаком `$`.

Примеры представлены на рисунках 20-22.

```

2 # Создание фреймов
3 ?data.frame
4 fio<-c("Антонов А.В.", "Борисов С.Р.", "Волошин И.В.", "Федоров А.Е.")
5 fio
6 country<-c("Россия", "Белоруссия", "Индия", "Китай")
7 country
8 capital<-c("Москва", "Минск", "Дели", "Пекин")
9 capital
10 popul<-c(146390000, 214600000, 1339180000, 1401296000)
11 popul
12 fr<-data.frame(FIO=fio, Coutry=country,
13               Capital=capital,
14               Population=popul,
15               check.rows = T,
16               check.names = T,
17               fix.empty.names = T)
18 fr

```

Рисунок 20 Пример создания фрейма

```

19 # Операции над фреймами
20 fr[,1]
21 fr[,2]
22 fr[3,]
23 fr[4,4]
24 fr[1,]
25 fr[2,]
26 fr$Population[1]
27 sum_popul<-fr$Population[3]+fr$Population[4]
28 sum_popul
29 sum_population<-sum(fr$Population)
30 sum_population

```

Рисунок 21 Операции над элементами фрейма

```

32 # Создание фрейма с пустыми значениями вектора
33 fam<-c(NA, NA, NA, NA)
34 fam
35 fr1<-data.frame(Family=fam, Coutry=country,
36                Capital=capital,
37                Population=popul,
38                check.rows = F)
39
40 fr1
41 fr1$Family[2]<-"Петрова О.Л."# добавление элемента в пустой список
42 fr1

```

Рисунок 22 Создание и обработка частично пустого фрейма

При обработке числовых данных фреймов полезной оказывается функция `transform()`:

`transform(x, tag1=value1, tag2=value2, ..., tag n=value n)`

Параметр `x` – объект, подлежащий трансформации. Параметры `tag1`, `tag2`, `tag n` определяют новые расчетные переменные в фрейме.

В примере, приведенном на рисунке 23, показано, как с помощью данной функции, не создавая новых объектов, можно, например, посчитать суммарные и средние баллы студентов за сессию.

```

2 fio<-c("Антонов А.В.", "Борисов С.Р.", "Волошин И.В.", "Федоров А.Е.")
3 fio
4 fr<-data.frame(fio,
5               matematik=c(4,2,3,4),
6               computer=c(4,3,3,5),
7               history=c(4,3,4,5),
8               economic=c(4,3,3,5))
9
10 fr
11 ?transform
12 dat<-transform(fr, sumx=matematik+computer+history+economic,
13               meanx=(matematik+computer+history+economic)/4)
14 dat|

```

Рисунок 23 Пример создания новых переменных в фрейме

Работа с датафреймами с использованием библиотеки dplyr

Библиотека dplyr – библиотека для удобной работы с датафреймами. С её помощью можно более быстро получать описание таблицы, сохранять полученные результаты и группировать данные по определенному признаку.

Установим библиотеку и обратимся к ней:

```
install.packages("dplyr")
```

```
library(dplyr)
```

Воспользуемся данными о качестве воздуха в Нью-Йорке, измеренными в период с май по сентябрь далекого 1973 года:

```
library(datasets)
```

Посмотрим на данные:

```
head(airquality)
```

Смысл понятен, но информацию о данных в более удобоваримом виде можно получить, используя функцию dplyr glimpse(). Команда транспонирует данные (поменяет местами строки и столбцы) и укажет количество наблюдений и переменных.

```
glimpse(airquality)
```

Основные функции dplyr и оператор %>%

Некоторые функции, встроенные в библиотеку, похожи на обычные функций, которые мы использовали на прошлом занятии. Например, функция select(), которая позволяет выбрать интересующие нас столбцы в датафрейме. Выберем только столбцы со скоростью ветра и днём.

```
select(airquality, Wind, Day)
```

Выберем все столбцы, имена которых содержат латинскую «o»

```
select(airquality, contains("o"))
```

Посмотрим на столбцы, имена которых начинаются с «Oz»

```
select(airquality, starts_with("Oz"))
```

Также с помощью `select()` можем исключить некоторые столбцы, которые нас не интересуют, поставив перед вектором столбцов минус (так же, как и раньше!):

```
select(airquality, -c(Wind, Day))
```

Столбцы можно выбирать по названиям, если столбцы идут подряд:

```
head(select(airquality, Ozone:Temp))
```

Если хотим отобрать интересующие нас наблюдения, нам потребуется другая функция – `filter()`. Например, выберем все наблюдения, когда температура поднималась выше 92 градусов по фаренгейту.

```
filter(airquality, Temp > 92)
```

Условия можно комбинировать как душе угодно. Вот хочется мне узнать, когда в июне значения измеренной солнечной радиации превышали 300 Лэнгли.

```
filter(airquality, Month == 6 & Solar.R > 300)
```

Рассмотрим еще одну команду `mutate()`, которая позволяет добавлять новые переменные (столбцы). Добавим столбец с температурой в цельсиях, который назовем `TempInC`:

```
mutate(airquality, TempInC = (Temp - 32) * 5 / 9)
```

Функция `summarise()` позволяет получить быстрое представление о всех данных в одном значении, например, посчитать среднее арифметическое. В комбинации с командой группировки данных `group_by()` можно легко находить интересующие значения для группированных данных. Простой пример: посчитаем среднюю температуру в каждом из месяцев.

```
summarise(group_by(airquality, Month),  
mean_monthly_temp=mean(Temp, na.rm = TRUE))
```

`na.rm = TRUE` просто не учитывает все отсутствующие данные при расчете среднего.

В рамках `summarise()` помимо нахождения среднего арифметического, можно использовать и другие команды, возвращающие скалярные значения, например `min()`, `max()`, `sum()`, `sd()`, `median()`.

Казалось бы, зачем использовать библиотеку `dplyr`, если результаты пока несильно отличаются от того, что мы делали на прошлом занятии без всяких библиотек. На самом деле, смысл использовать её есть.

В библиотеке `dplyr` есть особый оператор Pipe `%>%`, который позволяет выполнять операции пошагово. Оператор `%>%` позволяет совершать последовательные операции без сохранения промежуточных результатов.

Допустим, мы хотим увидеть, какая максимальная температура в градусах цельсия была зарегистрирована в летние месяцы.

```
airquality %>%  
  filter(Month == c(6,7,8)) %>% #оставляем только летние месяцы  
  mutate(TempInC = (Temp - 32) * 5 / 9)%>% # считаем температуру в  
градусах цельсия  
  group_by(Month) %>% # группируем по месяцам  
  summarise(max_temp_C = max(TempInC, na.rm = TRUE)) # находим  
максимальное значение температуры для каждого из летних месяцев
```

В библиотеке `dplyr` есть несколько других интересных и полезных функций. Например, `arrange()` – функция, которая сортирует таблицу в соответствии со значениями переменной (или переменных), расположенных по возрастанию (если переменная текстовая, то по алфавиту). Отсортируем, например, таблицу по показателю `Temp` сначала.

```
airquality %>% arrange(Temp) %>% head
```

А теперь на последние:

```
airquality %>% arrange(Temp) %>% tail
```

Задача. Выбрать строки в таблице, для которых значения `Ozon` не пустые (не `NA` в переменной `Ozon`), выбрать температуру больше 77, отсортировать строки по дню и посмотреть на итоговую таблицу.

Решение.

```
25 airquality %>%  
26   filter(!is.na(Ozone), Temp > 77) %>%  
27   arrange(Day) %>%  
28   view
```

Сначала мы выберем те строки в базе, для которых значения `Ozon` не пустые (`is.na()` и помним про отрицание – восклицательный знак), выберем температуру больше 77. Затем с помощью `arrange()` отсортируем строки по значениям `Day`. И, наконец, посмотрим на полученный датафрейм через `View`.

Экспорт данных из R в другие форматы

Обработанные в среде R данные могут быть сохранены во внешних файлах других форматов.

Для вывода данных в текстовые файлы с расширением `.txt` используется функция `write.tableQ`:

```
write.table(x, file = "", append = FALSE, quote = TRUE, sep = " ",  
  eol = "\n", na = "NA", dec = ".", row.names = TRUE,  
  col.names = TRUE, qmethod = c("escape", "double"),
```

```
fileEncoding = "")
```

Параметр `x` определяет объект данных R (вектор, матрицу, массив данных, фрейм), содержимое которого экспортируется во внешний текстовый файл.

Параметр `file` представляет собой строку, определяющую имя создаваемого при экспорте файла – в текущей папке R либо иной папке, полный путь к которой указан.

Значение остальных параметров во многом аналогично параметрам функции `read.table()`. Более точную информацию можно найти в справке.

Варианты использования функции `write.table()` приведены на рисунке.

```
2 # Экспорт данных в текстовый файл с помощью функции write.table()
3 ?write.table
4 fio<-c("Антонов А.В.", "Борисов С.Р.", "Волошин И.В.", "Федоров А.Е.")
5 fio
6 country<-c("Россия", "Белоруссия", "Индия", "Китай")
7 country
8 capital<-c("Москва", "Минск", "Дели", "Пекин")
9 capital
10 popul<-c(146390000, 214600000, 1339180000, 1401296000)
11 popul
12 fr<-data.frame(FIO=fio, Coutry=country,
13               Capital=capital,
14               Population=popul,
15               check.rows = T,
16               check.names = T,
17               fix.empty.names = T)
18 fr
19 # экспорт с номерами строк
20 write.table(fr, file = "my_fr.txt", append = FALSE, quote = TRUE, sep = " ",
21            eol = "\n", na = "NA", dec = ".", row.names = TRUE,
22            col.names = TRUE, qmethod = c("escape", "double"),
23            fileEncoding = "")
24 # экспорт без номеров строк
25 write.table(fr, file = "my_fr_1.txt", row.names = F)
```

Рисунок 24 Пример экспорта данных в текстовый файл

Для вывода данных из среды R во внешний файл MS Excel с расширением `*.csv` используется функция `write.csv()`:

```
write.csv(x, file = "")
```

Параметр `x` определяет объект экспортируемых данных. Параметр `file`, как и для функции `write.table()`, задает в виде строки имя создаваемого файла (в текущей папке или в указанном месте).

Варианты использования функции `write.csvQ` приведены на рисунке 25.

```

2 # Экспорт данных в файл Excel с помощью функции write.csv()
3 ?write.csv()
4 fio<-c("Антонов А. в.", "Борисов С.Р.", "Волошин И.В.", "Федоров А.Е.")
5 fio
6 country<-c("Россия", "Белоруссия", "Индия", "Китай")
7 country
8 capital<-c("Москва", "Минск", "Дели", "Пекин")
9 capital
10 popul<-c(146390000, 214600000, 1339180000, 1401296000)
11 popul
12 fr<-data.frame(FIO=fio, Coutry=country,
13               Capital=capital,
14               Population=popul,
15               check.rows = T,
16               check.names = T,
17               fix.empty.names = T)
18 fr
19 write.csv(fr, file="my_fr_2.csv")

```

Рисунок 25 Экспорт данных в файл MS Excel

Для экспорта таблиц данных (фреймов) из среды R во внешний файл MS Excel с расширением *.xlsx используется функция write.xlsx():

```
write.xlsx(x, file, sheetName="Sheet1",
          col.names=TRUE, row.names=TRUE, append= FALSE, showNA=TRUE)
```

Позиционный параметр x указывает на имя фрейма, который предполагается вывести во внешний файл.

Параметр file представляет собой строку, которая содержит имя создаваемого файла. Если приведено неполное имя без указания пути к файлу, создаваемый файл сохраняется в текущей рабочей папке.

Параметр sheetName определяет имя листа книги Excel в создаваемом файле.

Параметр col.names при значении TRUE предписывает выводить в файл имена столбцов данных; значение FALSE – исключает вывод.

Параметр row.names при значении TRUE выводит в файл имена строки в виде чисел, заданных в текстовом формате. Предпочтительней выглядит значение параметра FALSE.

Параметр append при значении FALSE указывает на замену созданного листа, если функция выполняется повторно. Значение TRUE, если указано новое имя листа, обеспечивает добавление в книгу нового листа, В противном случае формируется сообщение об ошибке.

Параметр showNA позволяет при значении TRUE выводить в файл пустые ячейки.

Прежде чем приводить пример использования функции write.xlsx() для экспорта полученных результатов из среды R а среду табличного процессора MS Excel, рассмотрим ряд функций, которые находят применение

перед экспортом с целью представления результатов в более читаемом варианте,

Функция `formatC()` служит для форматирования числовых данных и превращения результата в строки:

```
formatC(x, digits = NULL, width = NULL, format = NULL, flag = "",
mode = NULL, big.mark = "", big.interval = 3L,
small.mark = "", small.interval = 5L,
decimal.mark = getOption("OutDec"),
preserve.width = "individual", zero.print = NULL,
dropOtrailing = FALSE)
```

Основные параметры функции:

`x` – число или вектор вещественных чисел, которые предполагается отформатировать:

`digits` – количество знаков в дробной части числа;

`format` – форма представления числа: "f" – с фиксированной запятой, "e" – с плавающей запятой,

Значения других параметров можно узнать, вызвав справку:

?formatC либо help(formatC),

Функция `round()` используется для округления числовых данных;

```
round(x, digits = 0)
```

Первый параметр функции `x` представляет собой исходное число.

Параметр `digits` задает количество знаков после запятой.

Пример

Экспорт данных – фрейма в файл MS EXCEL с расширением `.xlsx`.

Из таблицы данных `зарплата.csv` создать с помощью функции `data.frame()` фрейм `newtable`, содержащий названия регионов ЦФО и средние заработные платы работников указанных регионов за 2008 и 2017 годы. Полученные числовые значения отформатировать. Содержимое фрейма экспортировать в файл MS Excel с расширением `*.xlsx` на указанный лист.

Добавить в скрипт команду на вывод значения фрейма на экран. Сравнить результаты вывода фрейма на экран и в файл MS Excel.

Решение


```

2 # Экспорт данных в файл Excel с помощью функции write.csv()
3 library("xlsx")
4 dat<-read.table("clipboard")#,h=F,dec = ".",sep = "")
5 dat
6 region<-dat[,1]
7 region
8 x2008<-dat[,2]
9 x2008
10 x2017<-dat[,11]
11 rost<-x2017/x2008
12 rost
13 x2017<-round(x2017.1)
14 rost<-round(rost,2)
15 rost
16 p<-getwd()
17 newtable<-data.frame(region,x2008,x2017,rost)
18 newtable
19 file<-paste(p,"newfile.xlsx",sep="/")
20 file
21 library(writexl)
22 write.xlsx(newtable,
23           file,
24           sheetName=sheet_user,
25           row.names=F,
26           colnames=T,
27           append=F)

```

Рисунок 26 Протокол скрипта

Списки (lists)

Список представляет собой «вектор векторов» в терминах R. Для тех, кто знаком с программированием, может показаться, что списки похожи на массивы. Это так, но списки, в отличие от массивов, могут содержать элементы разных типов. Например, в списке в R может быть сохранен вектор имен студентов (текстовый, тип *character*) и вектор их оценок (целочисленный, тип *integer*).

Для тех, кто знаком с Python: списки в Python и списки в R похожи. Списки в R – это вложенные списки в Python.

Списком называется многомерная упорядоченная структура разнотипных данных.

Для создания списков используется функция `list()`:

```
list(x1,x2,...,xn)
```

Параметры `x1,x2,...,xn` могут быть векторами, матрицами, массивами данных, фреймами и другими списками.

При создании списков обычно предварительно создаются его компоненты, впоследствии объединяемые с помощью функции `list()` в списки.

Пример списка с числовыми значениями:

```
L <- list(c(1, 2, 3, 4), c(5, 6, 7, 8))
```

```
L
```

```
## [[1]]
## [1] 1 2 3 4
##
## [[2]]
## [1] 5 6 7 8
```

А вот пример списка с элементами разных типов:

```
grades <- list(c("Ann", "Sam", "Tom"), c(8, 7, 5))
grades
## [[1]]
## [1] "Ann" "Sam" "Tom"
##
## [[2]]
## [1] 8 7 5
```

Так как в списках может храниться большое число разных векторов, для удобства им можно давать названия. Список `grades` можно было записать и так:

```
grades <- list(names = c("Ann", "Sam", "Tom"), marks = c(8, 7, 5))
```

И тогда отдельные вектора из списка можно было бы вызывать удобным образом:

```
grades$names # имена
## [1] "Ann" "Sam" "Tom"
grades$marks # оценки
## [1] 8 7 5
```

И если мы бы запросили у R структуру этого списка, мы бы увидели названия векторов, которые в него входят.

```
str(grades)
## List of 2
## $ names: chr [1:3] "Ann" "Sam" "Tom"
## $ marks: num [1:3] 8 7 5
```

Можно подумать: зачем нужно знать про списки, если на практике мы обычно будем сталкиваться с другими объектами – датафреймами (таблицами)? На самом деле, со списками мы тоже будем встречаться. Многие статистические функции выдают результат в виде списков. Когда результаты выводятся на экран, это не всегда заметно, но если мы захотим заглянуть внутрь, то увидим, что та же регрессионная выдача представляет собой объект, похожий на список, из которого можно выбрать вектор коэффици-

ентов (coefficients), вектор остатков (residuals), предсказанных значений (fitted.values) и так далее.

А как обращаться к элементам списка, если вектора в нем никак не названы?

Для обращения к элементам списка необходимо использовать двойные квадратные скобки:

```
L
## [[1]]
## [1] 1 2 3 4
##
## [[2]]
## [1] 5 6 7 8
L[[1]] # первый элемент списка, вектор (1, 2, 3, 4)
## [1] 1 2 3 4
```

Если нужно обратиться к «элементу элемента» списка (например, к числу 8 в этом примере), нужно сначала указать номер вектора, в котором находится элемент, а затем номер самого элемента в этом векторе.

```
L[[2]][4] # 8 – 4-ый элемент 2-ого вектора в списке
## [1] 8
```

Можно заметить, что список похож на матрицу: для того, чтобы обратиться к элементу, нужно указать «строку» (вектор) и «столбец» (положение в векторе).

Для того, чтобы добавить элемент в список, нужно четко понимать положение элемента в этом списке: будет ли это элементом самого списка или «элементом элемента»:

```
L
## [[1]]
## [1] 1 2 3 4
##
## [[2]]
## [1] 5 6 7 8
L[[3]] <- c(8, 9) # добавили в список третий вектор
L
## [[1]]
## [1] 1 2 3 4
##
## [[2]]
```

```
## [1] 5 6 7 8
##
## [[3]]
## [1] 8 9
L[[3]][3] <- 0 # добавили третий элемент третьего вектора в списке
L
## [[1]]
## [1] 1 2 3 4
##
## [[2]]
## [1] 5 6 7 8
##
## [[3]]
## [1] 8 9 0
```

Аналогичным образом можно изменять элементы списка:

```
L[[1]][1] = 99 # заменим 1 элемент 1 вектора в массиве на 99
L
## [[1]]
## [1] 99 2 3 4
##
## [[2]]
## [1] 5 6 7 8
##
## [[3]]
## [1] 8 9 0
```

Если в списке всего один элемент, при необходимости его можно быстро превратить в обычный вектор с помощью `unlist()`:

```
small_L <- list(c("a", "b", "c"))
small_L
## [[1]]
## [1] "a" "b" "c"
small_vec <- unlist(small_L)
small_vec
## [1] "a" "b" "c"
```

То же можно делать и со списками из нескольких векторов, тогда все склеится в один длинный вектор:

```
L
```

```
## [[1]]
## [1] 99 2 3 4
##
## [[2]]
## [1] 5 6 7 8
##
## [[3]]
## [1] 8 9 0
unlist(L)
## [1] 99 2 3 4 5 6 7 8 8 9 0
```

Преобразование типов и структур данных

В ходе обработки данных иногда приходится выполнять преобразования данных. Возможности преобразования представлены на рисунке.

```
2 # Преобразования типов данных
3 a<-1:10
4 a
5 # Преобразование вектора a в строковый вектор
6 a_str<-as.character(a)
7 a_str
8 # Конвертация строкового вектора в целочисленный вектор
9 d<-c("1","2","3")
10 d
11 d_int<-as.integer(d)
12 d_int
13 # Конвертация целочисленного вектора в номинальную
14 # шкалу (factor) со значениями 0 и 1
15 b<-c(1,1,0,0,1,1,0)
16 b
17 b_factor<-as.factor(b)
18 b_factor
```

Рисунок 27 Выполнение операций преобразования данных

Самостоятельная работа №3

Часть 1

1. Создайте из векторов из задачи 3 список (list) и назовите его info.

```
names <- c("Jane", "Michael", "Mary", "George")
```

```
ages <- c(8, 6, 28, 45)
```

```
gender <- c(0, 1, 0, 1)
```

- Выведите на экран имя Michael (обращаясь к элементам списка, конечно).
- Выведите на экран вектор gender.
- Назовите векторы в списке name, age, gender. Выведите на экран элементы вектора name.

- Добавьте в список вектор `drinks`, в котором сохранены значения: `juice, tea, rum, coffee`.

- Добавьте в список данные по еще одному человеку: `John, 2` года, мужской пол, любит молоко.

2. В R есть функция `strsplit()`, которая позволяет разбивать строки (текстовые переменные) на части по определенным символам.

Пусть у нас есть строка `s`:

```
s <- "a,b,c,d"
```

Мы хотим получить из нее вектор из 6 букв. Применяем функцию:

```
let <- strsplit(s, ",")
```

Получили почти то, что хотели. Почему почти? Потому что получили не вектор, а список!

```
class(let)
```

```
## [1] "list"
```

Превратим в вектор:

```
unlist(let)
```

```
## [1] "a" "b" "c" "d"
```

Теперь все в порядке, получили вектор из четырех элементов.

Дана строка `index`:

```
index <- "0,72;0,38;0,99;0,81;0,15;0,22;0,16;0,4;0,24"
```

Получите из этой строки числовой вектор `I`.

Часть 2

1. Поставьте библиотеку `randomNames`. Обратитесь к ней через `library()`.

2. Создайте вектор из 100 испанских имен:

```
set.seed(1234) # чтобы у всех получались одинаковые результаты
```

```
names <- randomNames(100, which.names = "first", ethnicity = 4)
```

3. Будем считать, что эти 100 имен – имена опрошенных респондентов.

Создайте вектор со значениями возраста респондентов:

```
ages <- sample(16:75, 100, replace = TRUE) # replace = TRUE – с повторяющимися значениями
```

А также вектор `polit` – политические взгляды респондентов:

```
views <- c("right", "left", "moderate", "indifferent")
```

```
polit <- sample(views, 100, replace = TRUE)
```

Создайте из полученных трёх векторов датафрейм.

4. Создайте столбец `id` с номерами респондентов.

5. Определите, сколько среди респондентов людей в возрасте от 25 до 30 лет (включительно). Определите, какую долю респондентов в нашей симпровизированной выборке составляют люди в возрасте от 25 до 30 лет. Выразите эту долю в процентах, округлите ее до 1 знака после запятой.
6. Создайте «факторный» вектор политических взглядов `polit_views`. Сколько у полученного фактора уровней? Добавьте в датафрейм столбец `polit_views`.

Часть 3

1. Загрузите файл `Firms.csv`. Почитать про базу можно здесь. Посмотрите на таблицу.
2. Сколько в датафрейме наблюдений? Сколько переменных? Какие это переменные?
3. Сколько в датафрейме полностью заполненных строк (наблюдений)? Выведите (если такие есть) наблюдения, содержащие пропущенные значения на экран.
4. Отфильтруйте наблюдения в таблице согласно следующим критериям:
 - а) фирмы с активами от 10000 до 20000 (включительно);
 - б) фирмы, число управляющих позиций, совместных с другими фирмами, которых не превышает 30;
 - с) фирмы транспортного сектора (TRN) под руководством управляющих из Канады (CAN);
5. Создайте переменную «натуральный логарифм активов» (`log_assets`) и добавьте её в датафрейм.
6. Постройте график, который может проиллюстрировать, какие паттерны пропущенных наблюдений можно зафиксировать в таблице.
7. Удалите пропущенные значения из базы данных.
8. Сохраните измененную базу данных в формате Stata (файл `"Firms.dta"`)

Часть 3

Импорт, предобработка и преобразование данных измерений

Выполнение данного задания в рамках изучения языка R должно сформировать целостное представление о потенциальных и явных возможностях обработки данных с использованием скриптового языка. В данной работе вам предстоит загрузить данные из открытого информационного источника, провести исследование загруженных данных, произвести пре-

образование данных к виду плоской таблицы, удобной для обработки, произвести некоторые вычисления по фрейму данных и его экспорт для возможной обработки в дальнейшем.

Необходимо загрузить данные из источника Интернет, поэтому убедитесь, что присутствует подключение к сети при загрузке данных.

Для выполнения задания понадобятся такие пакеты для обработки данных как “dplyr”, “tidyr”, “stringr”. Эти пакеты позволяют удобно преобразовывать данные к виду, который вам необходим. Для их установки выполните следующие действия:

1. Убедитесь, что папка пользователя не содержит кириллических символов или знаков препинания
2. Если содержит, убедитесь в том, что папка, в которую установлен R: “R/R-.../library” доступна для записи программными приложениями. Для этого необходимо открыть свойства папки и разрешить доступ на запись к данной папке.
3. Установите данные пакеты с помощью команды:

```
install.packages(c("dplyr", "readr", "stringr"))
```

Задание 1.

1. Загрузить при помощи функции read.csv() данные о положительных тестах на Covid19 из открытого репозитория github: https://raw.githubusercontent.com/CSSEGISandData/COVID-19/master/csse_covid_19_data/csse_covid_19_time_series/time_series_covid19_confirmed_global.csv

2. Исследовать данные, произвести некоторые первоначальные действия по просмотру свойств фрейма данных: установить размерность таблицы, имена столбцов, типы данных столбцов (по возможности, определить структуру информации, с которой столкнулись)

3. Произвести слияние колонок, связанных со страной и регионом – источником информации, для унификации метаданных статистики (вам может помочь пакет tidyr).

4. Отдельно произвести создание фрейма данных с колонками:

5.

Таблица 4

Поля фрейма данных

| Страна/Регион | Широта | Долгота | Сумма заболевших | Среднее число заболевших | Стандартное отклонение числа заболевших |
|---------------|--------|---------|------------------|--------------------------|---|
|---------------|--------|---------|------------------|--------------------------|---|

6. С помощью библиотек с(“dplyr”, “tidyr”, “stringr”) и стандартных возможностей языка R произвести создание нового итогового фрейма данных со следующей структурой таблицы:

Таблица 5

Формат итогового фрейма данных

| Дата <date> | Страна1 <int> | Страна2 <int> | ... | СтранаN <int> |
|----------------|------------------|------------------|-----|------------------|
| YYYY-MM-DD | X ₁₁ | X ₁₂ | ... | X _{1N} |
| YYYY-MM-DD | X ₂₁ | X ₂₂ | ... | X _{2N} |
| ... | ... | ... | ... | ... |
| YYYY-MM-DD | X _{M1} | X _{M2} | ... | X _{MN} |

В столбце Дата (date) должны храниться значения в стандартном для R типе данных date. Формат YYYY-MM-DD явно указывает на положение года, месяца, дня и разделителя между ними. X – это явное число подтвержденных положительных тестов на Covid19 на данный день в данной стране. Под названиями колонок “Страна1”, ... “СтранаN” подразумеваются те имена стран и регионов, которые вами предварительно созданы в пункте 3.

Для решения данной задачи вам возможно понадобятся функции: gsub(), ISOdate(), as.Date(), stringr::str_extract().

7. Итоговый фрейм данных, полученный в пункте 5, необходимо экспортировать в формат txt, csv, xlsx в поддиректорию вашего проекта с названием “data_output”.

Важно. Директория “data_output” должна создаваться во время выполнения скрипта, и не должна пересоздаваться во время повторного выполнения скрипта. Файлы с таблицей данных каждый раз должны перезаписываться в данной директории.

Контрольные вопросы

1. Каковы особенности фреймов
2. Приведите примеры создания фреймов
3. Приведите примеры экспорта данных из среды R во внешние файлы
4. Что собой представляют списки. Каким образом может быть создан список
5. Приведите примеры преобразования данных

Практическая работа №4

Ввод-вывод данных

Данные на обработку средствами R можно непосредственно задать и программе, используя, например, функцию `c()` или ввести из различных внешних источников:

- текстовых файлов;
- файлов, сохраненных в различных форматах табличного процессора MS Excel;
- специализированных статистических программ (IBM SPSS Statistics, Statistica, STATA и др.);
- баз данных (MS Access, Oracle и др.).

Второй вариант является основным, поскольку позволяет провести исследование сверхобъемных массивов информации, непосредственное создание которых в R-программе крайне затруднительно, да и нерационально, тем более, если есть возможность использовать данные, сформированные в ходе обработки другими программами,

Отметим некоторые особенности импортирования данных из внешних источников:

- в импортируемой таблице не должно быть пустых ячеек; если какие-то значения по тем или иным причинам отсутствуют, вместо них следует ввести NA (нет данных);
- в качестве первой строки в импортируемой таблице рекомендуется ввести заголовки столбцов-переменных; если данная строка отсутствует, то об этом следует указать при описании параметров функции, с помощью которой будет выполняться импорт внешнего последующие строки импортируемой таблицы в качестве первого элемента могут содержать заголовки строк, после которых должны следовать значения переменных;
- в именах столбцов таблицы и заголовках строк не допускаются пробелы; имена обязательно должны начинаться с буквы;
- во избежание проблем, связанных с кодировкой, текстовые элементы в импортируемых файлах рекомендуется создавать с использованием букв *латинского* алфавита;
- подлежащий импортированию файл рекомендуется поместить в текущую папку программы, установленную функцией `setwd()`;
- для облегчения выполнения импорта из внешнего файла

рекомендуется преобразовать импортируемую таблицу с данными в простой текстовый файл с расширением .txt или .csv (хотя это не обязательно).

Вывод результатов выполнения R-программы также может быть осуществлен различным образом:

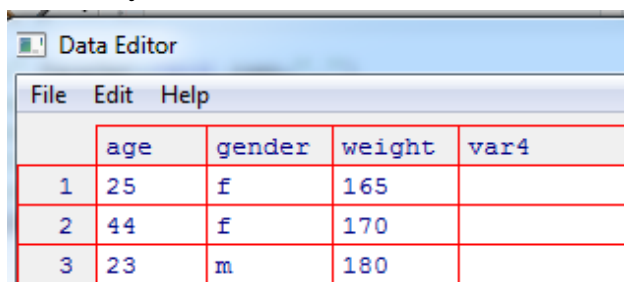
- на экран;
- на печать;
- в файлы различных форматов.

Импорт данных из текстовых файлов

Самый простой способ введения данных – это ввод с клавиатуры. Для этого необходимо создать пустую таблицу данных (или матрицу), указав названия и типы переменных. Затем открывается текстовый редактор функцией `edit()` с этим объектом, вводятся данные и сохраняется результат в виде объекта с данными.

В приведенном ниже примере создается таблица данных с названием `mydata` с тремя переменными: `age` (возраст, числовая), `gender` (пол, текстовая) и `weight` (вес, числовая). Затем откроется текстовый редактор, вносятся данные и сохраняется результат.

```
> mydata <- data.frame(age=numeric(0),  
  +gender=character(0), weight=numeric(0))  
> mydata <- edit(mydata)
```



| | age | gender | weight | var4 |
|---|-----|--------|--------|------|
| 1 | 25 | f | 165 | |
| 2 | 44 | f | 170 | |
| 3 | 23 | m | 180 | |

Рисунок 28 Копия объекта `mydata` открытая в редакторе данных функцией `edit()`

Для импорта данных из внешних источников, представляющих собой текстовые файлы, применяют функции `read.table()`, `read.csv()` и др. Рассмотрим форматы данных функций.

Функция `read.table()` обеспечивает считывание таблицы данных из внешнего источника – текстового файла:

```
read.table(file, header = FALSE, sep = "", quote = "\"",  
  dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),  
  row.names, col.names, as.is = ! stringsAsFactors,
```

```
na.strings = "NA", colClasses = NA, nrow = -1,
skip = 0, check.names = TRUE, fill = !blank.lines.skip,
strip.white = FALSE, blank.lines.skip = TRUE,
comment.char = "#",
allowEscapes = FALSE, flush = FALSE,
stringsAsFactors = default.stringsAsFactors(),
fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)
```

Раскроем назначение основных параметров.

Первый позиционный параметр `file` представляет собой строку, содержащую полный путь и импортируемому файлу, либо является собственным именем файла, находящимся в текущей папке.

Ключевой параметр `header` может принимать значение `TRUE` (истина) или `FALSE` (ложь, по умолчанию). Если его значение истинно, следовательно, импортируемая таблица из внешнего файла содержит строку заголовков столбцов. Если заголовки столбцов представляют собой числа, к ним добавляется литера, чтобы имя начиналось с буквы.

Параметр `sep` позволяет задать разделитель между данными в строке импортируемой таблицы. Если значение `sep = ""`, разделителем является один или несколько пробелов, знак табуляции или возврат каретки.

Параметр `dec` определяет знак – разделитель целой и дробной части числа. По умолчанию разделителем является точка (`dec = "."`); для таблиц, подготовленных в России и большинстве европейских стран – запятая (`dec = ","`).

Значением параметра `row.names` является вектор, содержащий имена строк таблицы. Если в импортируемой таблице есть заголовок и первая строка содержит на одно поле меньше, чем количество столбцов, первый столбец во второй и последующих строках воспринимается, как имя строки. Если имена строк отсутствуют, строки нумеруются.

Параметр `col.names` указывает на вектор имен переменных. Если имена переменных не определены, по умолчанию они формируются из символа `V`, за которым следует номер столбца.

Параметр `fill` в случае истинного значения позволяет заполнять более короткие строки импортируемой таблицы пустыми полями.

Пример

Пусть имеется текстовый файл `Primer01.txt`, в котором содержится таблица с числовыми данными. Полный путь к файлу известен. Заголовки строк отсутствуют. Необходимо выполнить ввод числовых значений в

R-программу с одновременным выводом данных на экран.

Решение

Текст скрипта:

Ввод и вывод на экран числовой таблицы из текстового файла Primer01.txt

по полному пути к файлу

```
> chem<-read.table (file = "D:/R/data/Primer01.txt", sep=" " ,header = TRUE)
> chem
  year wheat barley oats rye corn
1 1980   5.9    4.4  4.1 3.8    1
2 1981   5.8    4.4  4.3 3.7    1
3 1982   6.2    4.9  4.4 4.1    2
4 1983   6.4    4.7  4.3 3.7    3
5 1984   7.7    5.6  4.9 4.7    3
```

Рисунок 29 Фрагмент таблицы вывода на экран по полному пути файла

Пример

Пусть имеется текстовый файл Primer01.txt, находящийся в текущей папке и в котором содержится таблица с числовыми данными. Заголовки строк отсутствуют. Необходимо выполнить ввод числовых значений в R-программу с одновременным выводом данных на экран.

Решение

Текст скрипта:

Ввод и вывод на экран числовой таблицы из текстового файла Primer01.txt , находящегося в текущей папке.

Протокол выполнения скрипта:

```
> setwd("D:/R/data")
> chem<-read.table (file = "Primer01.txt", sep=" " ,header = TRUE)
> chem
  year wheat barley oats rye corn
1 1980   5.9    4.4  4.1 3.8    1
2 1981   5.8    4.4  4.3 3.7    1
3 1982   6.2    4.9  4.4 4.1    2
4 1983   6.4    4.7  4.3 3.7    3
5 1984   7.7    5.6  4.9 4.7    3
```

Рисунок 30 Фрагмент вывода на экран числовой таблицы

Пример

Пусть в текстовом файле Primer01.txt требуется после ввода импортируемой таблицы в среду R выполнить замену разделителя целой и дробной части чисел с запятой на точку и сохранить полученный результат в таблице.

Протокол выполнения скрипта:

```
> chem<-read.table (file = "D:/R/data/Primer01.txt", sep="" ,header = TRUE, dec = ",")
> chem
  year wheat barley oats rye corn
1 1980   5.9   4.4  4.1 3.8    1
2 1981   5.8   4.4  4.3 3.7    1
3 1982   6.2   4.9  4.4 4.1    2
4 1983   6.4   4.7  4.3 3.7    3
5 1984   7.7   5.6  4.9 4.7    3
```

Рисунок 31 Фрагмент вывода на экран числовой таблицы с на точку

Применение буфера обмена при импорте данных

Если перед выполнением соответствующей операции, находясь в среде любой программы – Excel, Word, Access и др., скопировать в Буфер обмена операционной системы какие-то данные, например, таблицу, то легко и безошибочно эти данные будут перемещены в среду R.

Причем ошибки не возникнут, даже если в таблице используются буквы русского алфавита и имеются другие ограничения (например, целая часть от дробной отделяется запятой, а не точкой). Ошибки будут возникать только в случае пробелов в текстовых названиях и когда копируются данные в Буфер обмена из файлов *.csv.

Пример

Пусть имеется таблица MS Excel. В числовых данных дробная часть от целой отделена запятой. Требуется с использованием Буфера обмена ввести данную таблицу в среду R. Рассмотреть возможные варианты решения, в том числе, если таблица размещена в текстовом документе MS Word.

Решение

Первый вариант: в окне редактора RStudio наберем требуемые операторы:

```
> Data<-read.table("clipboard",h=TRUE,dec = ",",sep = "\t")
> Data
x15.26..14.84..0.871..5.763..3.312..2.221
1 14.88, 14.57, 0.8811, 5.554, 3.333, 1.018
2 14.29, 14.09, 0.905, 5.291, 3.337, 2.699
3 13.84, 13.94, 0.8955, 5.324, 3.379, 2.259
4 16.14, 14.99, 0.9034, 5.658, 3.562, 1.355
5 14.38, 14.21, 0.8951, 5.386, 3.312, 2.462
```

Рисунок 32 Протокол выполнения скрипта

Второй вариант: в окне редактора RStudio наберем требуемые операторы:

```
> Data<-read.table("clipboard",h=FALSE,dec = ",",sep = "\t")
> Data
```

| | v1 |
|---|---|
| 1 | 15.26, 14.84, 0.871, 5.763, 3.312, 2.221 |
| 2 | 14.88, 14.57, 0.8811, 5.554, 3.333, 1.018 |
| 3 | 14.29, 14.09, 0.905, 5.291, 3.337, 2.699 |
| 4 | 13.84, 13.94, 0.8955, 5.324, 3.379, 2.259 |
| 5 | 16.14, 14.99, 0.9034, 5.658, 3.562, 1.355 |
| 6 | 14.38, 14.21, 0.8951, 5.386, 3.312, 2.462 |

Рисунок 33 Протокол выполнения скрипта

Если исходная таблица представлена в текстовом документе MS Word (или в файле другого формата), то допустимы ее копирование а Буфер обмена и отправка на выполнение в окне редактора RStudio команд:

```
> read.table("clipboard")
      v1 v2 v3 v4 v5
1 1357 23 14 45 66
```

Рисунок 34 Протокол выполнения скрипта

```
> Data<-read.table("clipboard",h=FALSE,dec = ".",sep = "",skip=1)
> dim(Data)
[1] 10 7
> Data
```

| | v1 | v2 | v3 | v4 | v5 | v6 | v7 |
|----|------|------|------|------|------|------|------|
| 1 | 19,2 | 19,1 | 21,9 | 23,1 | 23,5 | 25,7 | 27,3 |
| 2 | 21,0 | 20,3 | 23,5 | 24,6 | 25,1 | 26,9 | 27,3 |
| 3 | 17,5 | 16,4 | 17,7 | 18,1 | 18,6 | 20,4 | 21,5 |
| 4 | 16,3 | 16,3 | 16,8 | 17,9 | 17,8 | 19,2 | 20,2 |
| 5 | 20,0 | 18,3 | 20,2 | 21,3 | 20,9 | 22,5 | 25,5 |
| 6 | 20,6 | 20,4 | 22,4 | 23,6 | 23,7 | 26,1 | 26,4 |
| 7 | 28,3 | 27,7 | 29,8 | 30,8 | 31,0 | 33,4 | 37,5 |
| 8 | 20,7 | 20,4 | 22,6 | 23,9 | 24,4 | 25,9 | 26,8 |
| 9 | 16,6 | 16,7 | 18,1 | 19,5 | 19,8 | 22,2 | 24,7 |
| 10 | 14,7 | 14,8 | 16,5 | 17,0 | 16,5 | 18,4 | 20,1 |

```
> Data[2,]
      v1 v2 v3 v4 v5 v6 v7
2 21,0 20,3 23,5 24,6 25,1 26,9 27,3
> Data[1,3]
[1] 21,9
Levels: 16,5 16,8 17,7 18,1 20,2 21,9 22,4 22,6 23,5 29,8
>
```

Рисунок 35 Протокол выполнения скрипта

Анализ протоколов импорта табличных данных с использованием Буфера обмена показывает, что данный вариант является наиболее приемлемым, так как позволяет работать с каждым элементом введенной таблицы за счет сохранения результатов импортирования в фреймовом объекте.

Импорт из EXCEL-файлов

Следует отметить, что целесообразнее перед запуском RStudio запускать загрузчик RGui. Такой сценарий обеспечит более широкие возможности по включению в разрабатываемые R-скрипты инструментария различных дополнительных пакетов из многочисленных внешних источников хранения – так называемых зеркал, физически расположенных в

разных уголках мира.

Если предполагается в R-скрипте непосредственная обработка файлов, подготовленных в MS Excel (с расширением .xlsx для версий табличного процессора, начиная с 2007 года, или .xls – для более ранних версий), требуется предварительно скачать и установить пакет Ms*.

Перед установкой пакета xlsx, а также некоторых других пакетов из репозитория R, следует убедиться, что на компьютере установлены язык и вычислительная платформа Java, работающие на любой архитектуре – от стационарных компьютеров до мобильных устройств, и используемые при выполнении различных онлайн-приложений, обеспечивая их функциональность, быстродействие, безопасность и надежность.

Бесплатно загрузить Java можно, например, с сайта <https://www.java.com/ru/download/>, выбрав соответствующую операционную систему.

Скачать и установить пакет xlsx можно как в окне RStudio, последовательно выполнив команды `install.packages("xlsx")` и `library(xlsx)`, либо в окне загрузчика RGui. Последний вариант является более предпочтительным, так как позволяет избежать ситуаций, когда отсутствует доступ к репозиторию облачного хранилища CRAN (этому могут быть разные причины), из-за чего могут быть получены некорректные результаты инсталляции.

Рассмотрим последовательность действий в окне среды RStudio.

Вначале необходимо определить зеркало, из которого будет выбираться и скачиваться нужный пакет:

Кнопки Packages / Install.

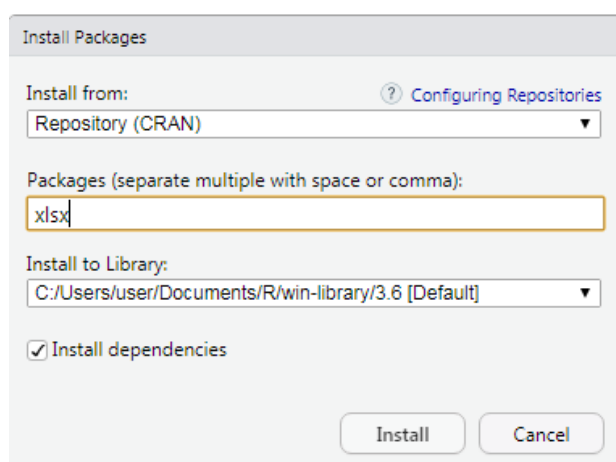


Рисунок 36 Пример скачивания и установки пакета xlsx

Начнется процесс инсталляции.


```
package 'xlsx' successfully unpacked and MD5 sums checked  
The downloaded binary packages are in  
C:\Users\user\AppData\Local\Temp\Rtmpw0BZs3\downloaded_packages  
> |
```

Рисунок 37 Завершение процесса инсталляции пакета xlsx

После рассмотрения и выполнения на компьютере указанных предварительных операций перейдем к описанию применения возможностей пакета xlsx.

Для непосредственного импорта в среду R нужных листов из Excel-файлов с расширениями .xlsx либо .xls используется функция read.xlsx():

```
read.xlsx(  
  xlsxFile,  
  sheet = 1,  
  startRow = 1,  
  colNames = TRUE,  
  rowNames = FALSE,  
  detectDates = FALSE,  
  skipEmptyRows = TRUE,  
  skipEmptyCols = TRUE,  
  rows = NULL,  
  cols = NULL,  
  check.names = FALSE,  
  sep.names = ".",  
  namedRegion = NULL,  
  na.strings = "NA",  
  fillMergedCells = FALSE  
)
```

Позиционный параметр xlsxFile представляет собой строковую константу либо переменную, содержащую полный путь к имени Excel файла или имя Excel-файла из рабочей директории, либо URL-адрес Excel-файла.

Назначение основных ключевых параметров:

sheetindex – задает имя или номер импортируемого листа Excel-файла;

startRow – определяет номер строки, начиная с которой будет осуществляться импорт данных; пустые строки в верхней части файла всегда пропускаются;

colNames – при значении TRUE устанавливает, что первая строка данных будет использоваться в качестве имен столбцов;

`rowNames` – при значении TRUE устанавливает, что первый столбец данных будет использоваться в качестве имен строк;

`detectDates` – при значении TRUE будет предпринята попытка распознать и преобразовать даты;

`skipEmptyCols` – при значении TRUE предписывает пропускать пустые столбцы;

`rows` – числовой вектор, содержащий номера строк, подлежащие импорту; при значении NULL импортируются все строки;

`cols` – числовой вектор, содержащий номера столбцов, подлежащие импорту; при значении NULL импортируются все столбцы;

`check.names` – при значении TRUE имена переменных во фрейме данных проверяются на предмет их синтаксической допустимости именам переменных.

В качестве примечания, напомним, что наличие в таблицах имен строк либо столбцов в русскоязычном варианте может вызвать из-за используемой на компьютере таблицы кодировки появление нечитаемых надписей. Во избежание такой проблемы следует в файлах использовать имена, набранные латиницей.

Пример

Пусть имеется Excel-файл `Зарплата_ЦФО.xls`, в котором на листе 2 содержится таблица. Известен полный путь к имени файла. Имена строк в таблице записаны латиницей, за исключением одного значения. Собственно таблица начинается с 4-й строки.

Используя функцию `readxlsx()`, выполнить ввод таблицы в среду R, присвоив ей значения таблице данных. Вывести результат импорта на экран.

В случае отсутствия RJava воспользоваться командами:

Установите 64 бит Java из <https://java.com/en/download/manual.jsp> .

Затем в windows cmd запустить

```
setx PATH "C:\Program Files\Java\jre1.8.0_211\bin\server;%PATH%"
```

(убедитесь, что ваш путь правильный).

Затем в RStudio запустить

```
Sys.setenv(JAVA_HOME="")
```

Протокол выполнения кода:

```

> Sys.setenv(JAVA_HOME="")
> library(rJava)
Предупреждение:
пакет 'rJava' был собран под R версии 3.6.3
> library(xlsx)
Предупреждение:
пакет 'xlsx' был собран под R версии 3.6.3
> getwd()
[1] "C:/Users/user/Documents"
> setwd("D:\\R\\data")
> read.xlsx("voenvuz.xls",sheetIndex = 1)
Ошибка в loadWorkbook(file, password = password) :
  Cannot find voenvuz.xls
> read.xlsx("voenvuz.xlsx",sheetIndex = 1)
  Name Age Height weight blood.group Rhesus.factor NA.
1   Ivan  23   178    80           2             +  <NA>
2  Peter  18   169    62           1             -  <NA>
3   Oleg  22   185    77           2             +  <NA>
4  Sergey 19   182    73           2             -  <NA>

```

Рисунок 38 Протокол выполнения скрипта

Импорт из файлов *.csv

При работе в R с файлами формата *.csv отрываются более широкие возможности представлен и я обрабатываемых текстовых данных на национальных языках. Кроме того, csv-файлы занимают существенно меньший объем, что снижает нагрузку на требования памяти, так как это фактически текстовые файлы, в которых данные разделяются запятой, а не пробелами или знаками табуляции, а дробная часть чисел отделяется от целой части точкой.

Файлы *.csv можно создать при работе в Excel, если при сохранении выбрать данный формат. Однако прежде чем сохранять Excel-файл в формате *.csv, необходимо в параметрах операционной системы задать необходимые изменения значений разделителей в числовых данных.

Для этого в операционной системе Windows 10 следует выполнить операции:

Кнопка Пуск → Кнопка Параметры → Группа Время и язык →

Ссылка Регион и язык → Ссылка Дополнительные параметры даты и времени, региональные параметры → Ссылка Изменение форматов даты, времени и чисел → Кнопка Дополнительные параметры..., → В окне Настройка формата на вкладке Числа (рис, 4.14) изменить разделитель целой и дробной части на точку, разделитель элементов списка на запятую → Кнопка ОК → Кнопка ОК

После указанных действий можно сохранить Excel-файл в формате *.csv. Для этого при сохранении файла следует указать его новый тип – CSV UTF-8 (разделитель – запятая) (*.csv).

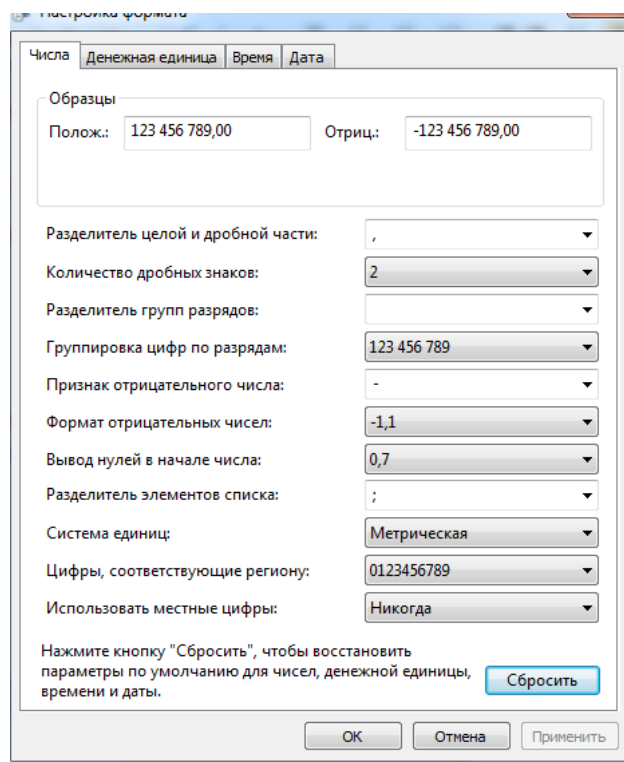


Рисунок 39 Окно изменения параметров разделителей числовых данных

| Файл | Правка | Формат | Вид | Справка |
|--|--------|--------|---------|---------------------|
| Регион, 20014,2015,2016,2017,2018,2019 | | | | |
| Белгородская область, | 15.26, | 14.84, | 0.871, | 5.763, 3.312, 2.221 |
| Брянская область, | 14.88, | 14.57, | 0.8811, | 5.554, 3.333, 1.018 |
| Владимирская область, | 14.29, | 14.09, | 0.905, | 5.291, 3.337, 2.699 |
| Воронежская область, | 13.84, | 13.94, | 0.8955, | 5.324, 3.379, 2.259 |
| Ивановская область, | 16.14, | 14.99, | 0.9034, | 5.658, 3.562, 1.355 |
| Калужская область, | 14.38, | 14.21, | 0.8951, | 5.386, 3.312, 2.462 |
| Костромская область, | 14.69, | 14.49, | 0.8799, | 5.563, 3.259, 3.586 |
| Курская область, | 14.11, | 14.1, | 0.8911, | 5.42, 3.302, 2.7 |
| Липецкая область, | 16.63, | 15.46, | 0.8747, | 6.053, 3.465, 2.04 |
| Московская область, | 16.44, | 15.25, | 0.888, | 5.884, 3.505, 1.969 |

Рисунок 40 Представление Excel-файла в формате *.csv, открытого в блокноте

Для импорта таблиц в среду R из файлов с расширением *.csv используется функция `read.csv()`:

```
read.csv(file, header = TRUE, sep = ",", quote="\"", dec = ".", fill = TRUE,
comment.char = "",...)
```

Назначение параметров данной функции во многом совпадает с назначением параметров функции `read.table()`.

Пример

Пусть имеется файл `test2.csv` (рис. 11), содержащий информацию о среднемесячной заработной плате работников всех отраслей по регионам Центрального федерального округа РФ за период с 2014 – 2019 гг. Полный путь к файлу известен. Имена строк записаны кириллицей без пробелов. Имена столбцов указаны годами.

Используя функцию `read.csv()`, выполнить ввод таблицы в среду R.

Сохранить таблицу в переменной-фрейме. Вывести различные варианты результатов ввода данных.

```
1 setwd("G:/R/data")
2 dat=read.csv("test2.csv",header=TRUE,sep=",")
3 dat# вывод данных таблицы
4 dat[4,5]#вывод данных Воронежской области в 2017 году
5 dat[3]#вывод данных в 2015 году
6 dat[2,]#вывод данных Брянской области по годам
```

Рисунок 41 Текст кода

```

      Регион x20014 x2015  x2016 x2017 x2018 x2019
1  Белгородская область  15.26 14.84 0.8710 5.763 3.312 2.221
2    Брянская область  14.88 14.57 0.8811 5.554 3.333 1.018
3  Владимирская область  14.29 14.09 0.9050 5.291 3.337 2.699
4  Воронежская область  13.84 13.94 0.8955 5.324 3.379 2.259
5   Ивановская область  16.14 14.99 0.9034 5.658 3.562 1.355
6   Калужская область  14.38 14.21 0.8951 5.386 3.312 2.462
7  Костромская область  14.69 14.49 0.8799 5.563 3.259 3.586
8    Курская область  14.11 14.10 0.8911 5.420 3.302 2.700
9   Липецкая область  16.63 15.46 0.8747 6.053 3.465 2.040
10  Московская область  16.44 15.25 0.8880 5.884 3.505 1.969
> dat[4,5]#вывод данных Воронежской области в 2017 году
[1] 5.324
> dat[3]#вывод данных в 2015 году
      x2015
1  14.84
2  14.57
3  14.09
4  13.94
5  14.99
6  14.21
7  14.49
8  14.10
9  15.46
10 15.25
> dat[2,]#вывод данных Брянской области по годам
      Регион x20014 x2015  x2016 x2017 x2018 x2019
2  Брянская область  14.88 14.57 0.8811 5.554 3.333 1.018

```

Рисунок 42 Протокол выполнения скрипта

Данные газодобычи

По ссылке, указанной ниже получить .xlsx файл с данными из открытого репозитория github:

<https://raw.githubusercontent.com/qwerty29544/RpracticeBook/master/2Data/01FlatTables/GAZ.csv>

Таблица EXCEL формата содержит данные о добыче нефти главной российской компанией добытчика данного вида ресурсов. Данные имеют следующую структуру:

1. Дата замера добычи со скважины.
2. Давление в трубе (МПа).
3. Температура установки (°C).
4. Добыча газа (кубометр в сутки).
5. Конденсат (кубометр в сутки).
6. Вода (кубометр в сутки).
7. ID скважины.

8. Куст скважины.
9. Группа скважины.

Задание 1.

1. Произвести импорт данных из файла формата EXCEL. Важно чтобы в результате экспорта все типы данных таблицы воспроизводились, как положено. Дата замера должна быть выражена через стандартный тип данных `date` в R.
2. Произвести очистку таблицы данных от строк с пустыми значениями признаков.
3. Перевести температуру установки в единицы измерения по Кельвину, удалив при этом первоначальный столбец температуры.
4. Преобразовать данные полей ID, Куст, Группа в ординальный формат данных.
5. Получить новые безразмерные поля, полученные на основе вычисления по старым полям:
 - отношение добычи газа к конденсату;
 - отношение добычи газа к добыче воды;
 - отношение добычи воды к добыче конденсата.
6. Отфильтровать данные измерений и получить выборку добычи на кустах за 2018 год.
7. Получить подвыборку данных измерений с куста по ID = 111
8. Вывести все ID скважин, добыча воды в которых никогда не превышала $2 \text{ м}^3/\text{сут.}$
9. Вывести ID скважин, суммарная добыча в день в которых не опускалась ниже $1000 \text{ м}^3/\text{сут.}$ Если данному условию удовлетворяют все, или не удовлетворяет ни одна из скважин, то подобрать значение суммарной добычи в день по кусту, в котором окажутся только 3 наименования.
10. Вывести название группы кустов, которая была самой результативной по добыче газа по результатам 2018 года.
11. Вывести название куста, который был самым результативным по добыче воды по результатам 2018 года.
12. Вывести название куста, в котором среднее отношение добычи нефти к добыче воды было самым наибольшим за всё время наблюдений.

Контрольные вопросы

1. Охарактеризуйте способы ввода данных из внешних источников
2. Каким образом можно ввести информацию в среду R из текстовых файлов

3. Каковы особенности представления файлов в формате .csv
4. Какие функции используются для импорта данных из текстовых файлов
5. Каковы особенности импорта данных из текстовых файлов
6. В чем преимущества импорта данных из Буфера обмена
7. Каковы особенности импорта данных из файлов Excel

Практическая работа №5

Программирование разветвлений и циклов

Организация разветвлений

Для организации разветвлений в программе на языке R можно использовать несколько операторов. Один из них – условный оператор if: ff {условие) {операторы!}

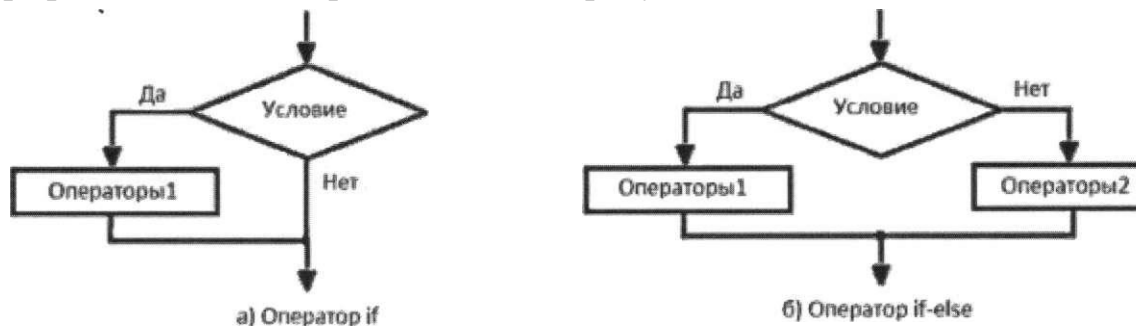
Данный оператор вначале проверяет логическое условие, В случае его истинности выполняются операторы!, указанные е фигурных скобках. Далее выполняется оператор, следующий за If. В случае ложности логического условия операторы в фигурных скобках не выполняются. Управление сразу передается к последующему за if оператору.

Другой оператор разветвления – if-else представляет более общий вариант первого оператора:

if (условие) {операторы1} else {операторы2}

После проверки логического условия в случае его истинности выполняется группа операторы1, в противном случае – операторы2. В дальнейшем управление передается оператору, следующему за if- else.

Графические схемы представлены на рисунке.



Еще один оператор, реализующий разветвления в программах на языке R, – оператор switch является множественным переключателем.

switch(условие, {операторы1},{ операторы2},...,{ операторы n})

Параметр условие может быть арифметическим или логическим выражением. В зависимости от полученных целочисленных результатов проверки арифметического условия switch обеспечивается выполнение одной из групп операторов из списка операторы1, операторы2,..., операторы n, после чего выполняется последующий за switch оператор.

Схема представлена на рисунке.



Рисунок 43 Схема выполнения оператора switch

Приведем ряд примеров на использование условных операторов.

Пример 10.

Пусть имеется вектор целочисленных значений. В зависимости от выполнения некоторого условия требуется с помощью оператора if изменить значение первого и остальных элементов исходного вектора.

Тест скрипта:

```

1  # пример 1
2  y<-1:10
3  y
4  if(y[1]==1)
5  {
6      y[1]<-y[1]+4
7      y[2:10]<-y[2:10]+6
8  }
9  y
10 # пример 2
11 y<-0
12 x<-3
13 switch (x,
14     {y<-y+86
15     "отлично"},
16     {y<-y+70
17     "хорошо"},
18     {y<-y+50
19     "удовлетворительно"},
20     {y<-y+0
21     "не удовлетворительно" })
22 y

```

Вариант использования оператора switch с логическим условием. Поскольку условие истинно, оператор выдает информацию обработки группы операторы1.

Тест скрипта:

```

23 # Пример 3
24 y<-0
25 x<-3
26 switch (x<5,
27   {y<-y+86
28     "отлично"},
29   {y<-y+70
30     "хорошо"},
31   {y<-y+50
32     "удовлетворительно"},
33   {y<-y+0
34     "не удовлетворительно" })
35 y

```

Организация циклов

Циклы в R могут быть организованы с помощью нескольких операторов, реализующих схемы как с фиксированным, так и с заранее неизвестным числом повторений, как с верхним, так и с нижним окончанием, определяющим условия завершения цикла.

Если число повторений тела цикла заранее известно, используют для организации циклов оператор `for`.

`for (условие) {операторы тела цикла}`

Параметр условие задает правила изменения переменной цикла, определяющей его повторение операторов тела цикла фиксированное число раз. По достижению заданного числа повторений цикл завершается. Алгоритмическая схема оператора `for` представлена на рисунке.



Рисунок 44 Схема выполнения цикла `for`

Если заранее неизвестно число повторений тела цикла, используют оператор `while`:

while (условие) {операторы тела цикла}

Истинное значение параметра условие предписывает выполнять операторы тела цикла. Завершение повторений наступает при нарушении выполнения условия. Если условие цикла изначально ложно, операторы тела цикла вовсе не будут выполняться. В связи с этим оператор while еще называют циклом с верхним окончанием.

Алгоритмическая схема оператора while приведена на рисунке.



Рисунок 45 Схема выполнения цикла while

Третий циклический оператор – repeat также обеспечивает бесконечное число повторений тела цикла:

repeat (условие) {операторы тела цикла}

Данный оператор выполняет тело цикла до тех пор, пока условие цикла не станет истинным. Если условие цикла изначально истинно, тело цикла выполнится один раз. Это цикл с нижним окончанием.

Алгоритмическая схема оператора repeat представлена на рисунке.



Рисунок 46 Схема выполнения цикла repeat

Во всех вариантах построения циклов может быть организован принудительный выход из циклического процесса с помощью оператора break. Однако такое завершение нарушает принятые соглашения о технологии программирования и структурированности программ, исключаящие в программе безусловные переходы.

Приведем несколько примеров на построение циклов.

Пример

Рассчитать с помощью оператора for сумму и произведение чисел от 1 до 10.

Решение

Текст скрипта:

```
2 s<-0
3 p<-1
4 n<-10
5 for (i in 1:n)
6 {
7   s<-s+1
8   p<-p*i
9 }
10 s
```

Пример.

Рассчитать с помощью оператора while сумму членов ряда:

$$y = x - \frac{x^3}{3} + \frac{x^5}{5} - \dots \frac{x^{(2n-1)}}{2n-1}, \text{ где } n - \text{номер члена ряда.}$$

В вычислениях использовать ограничение на величину очередного члена ряда: $|a_n| > 0,00001$.

Решение.

Текст скрипта:

```
13 x<-0.5
14 a<-x
15 y<-a
16 n<-1
17 while (abs(a)>0.00001)
18 {
19   print(n)
20   print(a)
21   print(y)
22   n<-n+1
23   a<-a*x^2/(2*n-1)
24   y<-y+a
25 }
```

При использовании оператора repeat результаты могут оказаться некорректными в силу того, что проверка выхода из цикла осуществляется после выполнения тела цикла. Рекомендуется проанализировать приве-

денный ниже простейший скрипт вывода оценки студента на экзамене, выполненный с помощью оператора repeat.

В данный скрипт целенаправленно была введена ошибка, чтобы показать, что тело цикла выполняется всегда хотя бы один раз, даже если изначально условием предписано цикл не выполнять.

Решение.

Текст скрипта:

```
27 v<-c("привет! ваш балл за экзамен -")
28 ball<-2
29 repeat
30 { print(v)
31   print(ball)
32   ball<-ball+1
33   if(ball<=3)
34   {
35     break
36   }
37 }
```

Самостоятельная работа №5

Часть 1

1. Напишите код, который запрашивает у пользователя его имя и фамилию (отдельные строки “Enter your name:” и “Enter your surname:”), сохраняет их и выводит на экран “Hello, [name] [surname]! Welcome to R!”
2. Напишите код, который сохраняет число, которое ввел пользователь с клавиатуры (предполагается, что пользователь вводит только числа, причем в правильном формате – в качестве разделителя использует точку), в переменную x и если x является целым числом, то выводит на экран сообщение “It is an integer.”, а если x не является таковым, то выводит “It is not an integer”.
3. Напишите код, который запрашивает у пользователя число элементов вектора (“Enter number of elements:”), сохраняет его в переменную n и создает вектор заданной длины, состоящий из пропущенных значений.
4. Напишите код, который запрашивает у пользователя число элементов вектора (“Enter number of elements:”), сохраняет его в переменную n и создает вектор заданной длины, состоящий из пропущенных значений. Дальше, если индекс элемента четный, то этот элемент заменяется на 1, если нечетный – на 0.
5. Дан вектор оценок студентов grades10 (оценки указаны в 10-балльной шкале). Напишите код, который на основе вектора grades10 создает вектор grades5 – вектор оценок в 5-балльной шкале:

- 0 – это 1
 - менее 4 – это 2
 - [4, 5] – это 3
 - [6, 7] – это 4
 - [8, 10] – это 5
6. Напишите код, который запрашивает у пользователя размерность матрицы (предполагается, что пользователь вводит число строк и столбцов через пробел), сохраняет их и создает единичную матрицу заданной размерности. Если невозможно создать единичную матрицу заданной размерности, на экран выводится сообщение “Impossible to create an identity matrix with such dimensions”.

Подсказка: единичная матрица – квадратная матрица (число строк равно числу столбцов), на главной диагонали которой стоят 1, а все остальные элементы равны 0.

Пример:

[1 0 0]

[0 1 0]

[0 0 1]

7. Дан вектор стран:

```
cnt <- c('France', 'Hungary', 'Ukraine', 'Romania', 'Germany', 'Russia', 'Finland', 'Italy', 'Spain')
```

Напишите код, который перебирает элементы вектора cnt до тех пор, пока не дойдет до России. Если название страны отлично от ‘Russia’, на экран должно выводиться сообщение “It is not Russia.”, а если название совпадает с ‘Russia’, на экран должно выводиться сообщение: “Russia is found. Its number is n”, где n – порядковый номер России в списке.

а) Используйте цикл for.

б) Используйте цикл while.

Внимание: Ваш код должен работать корректно и в том случае, если мы изменим порядок элементов в векторе cnt!

Часть 2

Поскольку R является языком, широко использующим математические конструкции в качестве базовых правил, посмотрим, насколько хорошо R справляется со стандартными задачами высшей математики. В данной работе необходимо создать скрипт, который производить вычисление площади под графиком функции с помощью одного из четырех численных методов, выбранных пользователем в процессе выполнения скрипта:

График функции:

$$y(x) = (x^3 - x^2 + 5x + 5\sin(2x + x^2) * \cos(2x + x^2) + 3) * e^{-x} \quad (1)$$

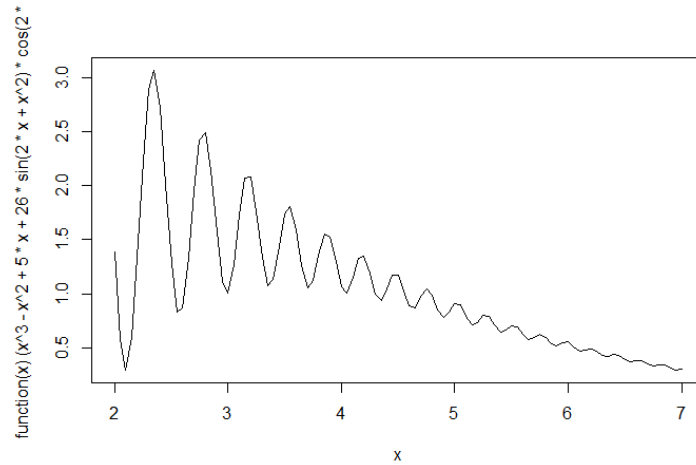


Рисунок 47 График изучаемой функции

Численные методы:

1. Метод правых прямоугольников:

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n-1} (f(a + h * i) * h), \quad (2)$$

где:

- $h = \frac{b-a}{n}$
- $x_0 = a$

2. Метод срединных прямоугольников:

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n-1} \left(f\left(a + h * i + \frac{h}{2}\right) * h \right), \quad (3)$$

где:

- $h = \frac{b-a}{n}$
- $x_0 = a$

3. Метод трапеций:

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n-1} \left(f(a + h * i) + f(a + h * (i + 1)) \right) * \frac{h}{2}, \quad (4)$$

где:

- $h = \frac{b-a}{n}$
- $x_0 = a$

4. Метод Симпсона:

$$\int_a^b f(x)dx \approx \frac{h}{3} \left(f(a) + f(b) + 4 \sum_{i=1}^n f(x_{2i-1}) + 2 \sum_{i=1}^n f(x_{2i}) \right), \quad (5)$$

где:

- $h = \frac{b-a}{2n}$
- $x_0 = a$
- $x_i = x_{i-1} + h$
- $i = 1, 2, \dots, 2n$

Задание 1.

1. Создать скрипт на языке R, считывающий данные пользователя с клавиатуры или пользовательского файла (структуру ввода информации из файла и саму структуру файла определяет разработчик). Данные, вводимые пользователем, состоят из следующих полей:

- Левая граница интегрирования (точка $a = x_0$ на оси абсцисс).
- Правая граница интегрирования (точка $b = x_n$ на оси абсцисс).
- Количество разбиений (n – число дискретных отрезков суммирования).
- Метод (кодировка метода остаётся на усмотрение разработчика).

В случае с пользовательским вводом данных численного эксперимента с клавиатуры необходимо в скрипте реализовать цикл, который будет выполнять эксперименты вплоть до того, как пользователь скажет о завершении работы программы.

В случае с вводом данных из файла данных, требуется исполнить весь список численных экспериментов или список всех строчек входных данных.

2. Скрипт должен выполнять интегрирование функции (1) одним из указанных в работе численных методов на выбор пользователя программы в рамках одного численного эксперимента

3. Результат каждого из численных экспериментов необходимо записывать в файл данных отдельной записью, состоящей из следующих полей:

- Уникальный номер эксперимента.
- Начальная граница интегрирования.
- Конечная граница интегрирования.
- Количество разбиений.
- Использованный метод.
- Полученный результат интегрирования.

В случае с пользовательским вводом, каждый ввод данных для эксперимента должен сопоставляться с одной строчкой результата численного вычисления интеграла в выходном файле вплоть до завершения работы программы.

В случае с вводом данных из файла данных, количество записей в выходной таблице данных должно равняться количеству записей в входной таблице данных.

Имя файла вывода задаётся пользователем при старте скрипта до расширения (расширение и формат файла вывода остаётся на определение программистом).

Путь к файлу ввода задаётся пользователем перед выполнением программы в случае выбора данной опции ввода.

Формат файла ввода должен быть определён разработчиком и детально описан перед сдачей работы. Основной перечень данных, который должен содержать файл, описан в пункте 1.

Пользователь должен иметь на старте скрипта выбор между пользовательским вводом с консоли и вводом данных из файла.

Выполнение задания засчитывается по выполнению всех поставленных в задаче условий и ограничений при реализации метода, а также по наличию рабочего скрипта, осуществляющего работу программы.

Часть 3

Итерационный метод

Тема данной работы является продолжением темы (самостоятельная работа №2). В работе предстоит усовершенствовать программу, связанную с реализацией итерационного метода, а именно метода простой итерации решения СЛАУ.

Метод простой итерации заключается в последовательном умножении преобразованной матрицы оператора на приближаемое решение. Таким образом, с каждой итерацией ожидаемая ошибка вычисления уменьшается, стремясь к относительно точному решению задачи.

Точного ответа на итерационных процедурах не достигается. Однако итерационные процедуры применяются в тех задачах, где важна скорость вычисления больших СЛАУ, например, в физико-математических задачах моделирования физических явлений. Вся скорость решения СЛАУ итерационными методами заключается в сложности вычисления.

Метод простой итерации имеет недостаток в большом числе задач, поскольку на данный метод накладывается сильное ограничение – спек-

тральный радиус матрицы оператора $\sigma(A)$ должен быть меньше единицы. В этом случае итерационная последовательность будет сходиться.

Алгоритм, определённый методом:

1. Получить исходные данные метода: матрица оператора A , вектор свободных членов в правой части f , вектор начальных значений u_0 , граничное количество итераций метода n_iter , требуемая точность вычисления eps .

2. Найти максимальное значение из элементов матрицы A и вектора свободных членов f , разделить все элементы матрицы и вектора на данное значение.

3. Вычислить матрицу B равную:

$$B = I - A, \quad (6)$$

где I – единичная матрица, A – исходная матрица оператора.

4. Сохранить в нулевой элемент итераций вектор нулевых значений u_0 , вычислить первую итерацию по схеме:

$$u_1 = B * u_0 + f, \quad (7)$$

умножение подразумеваем под матричным.

5. Обновить счётчик итераций, проверить критерий остановки по относительной ошибке через метрику:

$$\max(|\vec{u}_1 - \vec{u}_0|) > eps, \quad (8)$$

6. Если число итераций не больше положенного, или метрика не меньше заданного значения, то продолжаем итерации по схеме:

$$u_{i+1} = B * u_i + f, \quad (9)$$

7. По выходу из цикла возвращается последний вычисленный вектор u в качестве ответа на задачу.

Задание 1.

1. Прочитать документацию к функции `stopifnot()`.

2. Создать скрипт, который позволит решить СЛАУ методом простой итерации. Организовать ввод данных:

- размерности задачи (n – размер квадратной матрицы);
- матрицы A ;
- вектора свободных членов f ;
- количества граничных итераций;
- точность вычисления вектора u .

Выбор формата внесения данных выбирается программистом и должен быть детально описан при сдаче задания на проверку. Вектор начальных значений u_0 определяется с помощью функции `norm(n)`.

3. Вывести в скрипте ответ на задачу решения СЛАУ.

4. Протестировать скрипт на наличие слабых мест алгоритма – привести пример задачи решения СЛАУ, которая не решается методом простой итерации по причине расхождения метода (увеличение ошибки).

5. Работа метода должна быть защищена от случайного ввода путём обработки исключительных событий в `stopifnot()`, некоторые из них:

- ввод строки в матрицу оператора или вектор свободных членов;
- ввод отрицательного числа итераций;
- ввод отрицательной относительной ошибки;
- ввод отрицательного числа размерности задачи;
- ввод числа размерности меньше 2.

Подумать о других возможных исключительных случаях работы скрипта.

Контрольные вопросы

1. Какие операторы используются в R для организации ветвлений
2. Как в R могут быть организованы циклы
3. В чем суть циклов с верхним окончанием. Чем они отличаются от циклов с нижним окончанием
4. Сравните особенности циклов с фиксированным и заранее неизвестным числом повторений
5. Чем отличается цикл `repeat` от оператора цикла `while`

Практическая работа №6

Создание пользовательских функций

В языке R достаточно просто создавать пользовательские функции, реализующие определенный алгоритм обработки данных, с целью многократного вычисления результатов по заданным параметрам, его анализа, в том числе путем представления в графической форме.

Напомним, функцией в программировании называется программный модуль, имеющий один или несколько входных формальных параметров, используемых для реализации алгоритма функции, и возвращающий в точку вызова функции единственное вычисленное значение. Во многих языках программирования, кроме функций, имеются также подпрограммы, которые могут возвращать через аппарат параметров несколько вычисленных данных.

В языке R функции в некоторой степени могут выполнять роль как традиционных функций, так и традиционных подпрограмм. Правда, в последнем случае возвращаемые значения указываются в теле функции структурной переменной (вектором, матрицей, списком, фреймом) с использованием оператора, вернее функции `return()`.

Функции в R имеют имя, по этому имени осуществляется их вызов. Параметры функции записываются в круглых скобках после имени функции и разделяются друг от друга запятой. Различают позиционные параметры и ключевые.

При вызове функции значения позиционных параметров указываются в той же последовательности, как они были определены при объявлении (создании) функции. Ключевые параметры следуют после позиционных, их порядок несущественный, так как они указываются в форме:

имя_ключевого_параметра=значение

Имена ключевых параметров при вызове функции могут не указываться, достаточно просто привести в качестве фактических параметров их значения, однако в этом случае следует строго соблюдать ту последовательность параметров, которая была установлена при создании функции.

Если при создании функции для некоторых ключевых параметров были установлены значения по умолчанию и эти значения при вызове функции устраивают пользователя, их можно не указывать.

Значения параметров функции могут быть заданы именами объектов, литералами (константами), выражениями или с использованием других функций.

При обращении к функциям в R значения нечисловых параметров, как ключевых, так и позиционных, часто записывают в двойных кавычках,

В R есть функции, не имеющие параметров. При обращении к таким функциям круглые скобки по-прежнему обязательны.

Возвратимся к информации о создании пользовательских функций.

Заголовок функции имеет вид:

```
имя_функции<- function (формальные_параметры)
```

После заголовка функции следует ее тело. Оно обрамляется фигурными скобками. Открывающаяся скобка {определяет начало тела функции, закрывающаяся скобка} – его конец.

В теле функции могут указываться выражения и операторы, а также комментарии.

Если тело функции содержит только одно вычисляемое выражение, фигурные скобки можно не указывать.

Если предполагается, что функция должна возвращать несколько вычисленных значений, следует в теле функции сформировать структурную переменную, элементами которой должны быть найденные в теле функции вычисленные значения (определенные, например, с помощью оператора присваивания, функций формирования вектора `c()`, матрицы `matrix()` и т.п.). Результат, возвращаемый функцией, в таком случае должен указываться перед окончанием тела функции оператором `return` (структурная_переменная).

Подводя итоги, представим две схемы вызова функций из программного скрипта – когда функция возвращает одно значение или несколько. Схема вызова пользовательских функций показана на рисунке 45.



Рисунок 48 Схема вызова пользовательских функций с возвращением единственного (1) и нескольких значений (2)

При описании пользовательских функций можно определить значения параметров, задаваемые по умолчанию. Если все параметры функции заданы по умолчанию при ее создании, при обращении к такой функции фактические значения параметров в круглых скобках можно не указывать,

Пример

Разработать пользовательскую функцию, вычисляющую выражение:

$$f = c - x_1^\alpha - x_2^\alpha \quad (10)$$

Привести пример вызова разработанной пользовательской функции.

Решение

Текст скрипта:

```

2 # Разработка пользовательских функций
3 f<-function(x1,x2,c,a){
4   c-x1^a-x2^a
5
6 }
7 m<-f(2,1,1.2,2)
8 m
9
10 school <- function(a){
11   b = a+a*a
12   print(b)
13 }
14 school(2)
15 school(12)
16 school(22)

```

При описании пользовательских функций можно определить значения параметров, задаваемые по умолчанию. Если все параметры функции заданы по умолчанию при ее создании, при обращении к такой функции фактические значения параметров в круглых скобках можно не указывать.

Пример

Разработать пользовательскую функцию, вычисляющую выражение:

$$g = c - x^a - y^a \quad (11)$$

Привести пример вызова разработанной пользовательской функции.

Решение.

Текст скрипта:

```

18 g<-function(x=0.5,y=0.5,c=1,a=2){
19   c-x^a-y^a
20
21 }
22 r<-g()
23 r

```

Как было сказано, функции могут возвращать несколько значений.

Пример

Разработать пользовательскую функцию, вычисляющую сумму, произведение и разность двух чисел.

Привести пример вызова разработанной пользовательской функции.

Решение

Текст скрипта:

```

25 poly<-function(a,b)
26 {
27   sum<-a+b
28   prod<-a*b
29   razn<-a-b
30   y<-c(sum,prod,razn)
31 }
32 g1<-poly(3,7)
33 g1[1]
34 g1[2]
35 g1[3]

```

Самостоятельная работа

Часть 1

1. Напишите_функцию, которая принимает на вход базу данных, считает, сколько в базе данных переменных числового типа (numeric), сколько – факторного (factor) и сколько – текстового (character) и возвращает поименованный вектор, содержащий результаты подсчета. Если не знакомы с функциями в R (объект function), можно просто написать код, который выполняет действия, описанные выше.

Пример поименованного вектора: `res <- c("a" = 4, "b" = 5, "c" = 6).`

Дана база данных, с которой нужно поработать:

```
id <- 1:3
```

```
country <- c("Flatland", "Wonderland", "Sphereland")
```

```
craziness <- c(20, 15, 18)
```

```
region_type <- c("A", "B", "A")
```

```
author <- c("Abbot", "Carroll", "Burger")
```

```
size <- c(10, 100, 30)
```

```
m <- cbind(id, country, craziness, region_type, author, size)
```

```
df <- as.data.frame(m)
```

2. Напишите функцию, которая принимает на вход базу данных, выбирает из нее числовые переменные (столбцы типа numeric) и сохраняет их в новую базу данных. Если не знакомы с функциями в R (объект function), можно просто написать код, который выполняет действия, описанные выше.

3. Попробуйте выполнить задания 1-2 без использования apply-функций (если Вы их и не использовали, можете двигаться дальше).

4. Напишите функцию, которая принимает на вход вектор значений и, если вектор числовой, возвращает его медиану, если нет, выводит на экран сообщение "Vector is not numeric, cannot compute the median". Если не знакомы с функциями в R (объект function), можно просто написать код, который выполняет действия, описанные выше.

Часть 2

Некоторые методы обработки временных рядов

В данной работе мы рассмотрим некоторые методы обработки данных типа временного ряда и задачи, связанные с обработкой такого рода структур.

Для данной самостоятельной работы необходимо загрузить некоторые данные временного ряда цен акций компании с сайта finance.yahoo.com при помощи пакета “quantmod”. Это производится путём выполнения следующего скрипта:

```
if ("quantmod" %in% rownames(installed.packages()) == FALSE) {  
  install.packages("quantmod")  
}  
library(quantmod)  
  
if ("stringr" %in% rownames(installed.packages()) == FALSE) {  
  install.packages("stringr")  
}  
library(stringr)  
  
# Мы хотим загрузить акции с данными наименованиями в yahoo  
downloadable_stocks <- c("ATVI", "^IXIC")  
  
# Функция получения фреймов с данными  
quantmod::getSymbols(Symbols = downloadable_stocks,  
                      src = "yahoo",  
                      from = as.Date.character("1900-01-01"))  
  
# Функция get() позволяет получить содержимое объекта по его  
названию-строке  
# Мы можем и не знать названия акций в скрипте, но всё равно рабо-  
тать с ними  
# при пользовательском вводе названий  
df <- data.frame(get(downloadable_stocks[1]))  
# Применяем регулярное выражение для поиска и удаления ненужных  
символов  
downloadable_stocks <- stringr::str_remove(downloadable_stocks,  
"[:punct:~\\^]")  
# Удалим полученные объекты  
rm(list = downloadable_stocks)  
  
Теперь у нас имеется фрейм данных df, который содержит 6 колонок, в  
каждой из которых содержится временной ряд, связанный с ценами акций  
компании. В силах выполняющего это задание поменять на своё усмотрение
```

в скрипте переменную `downloadable_stocks` на любое другое значение наименования акций из существующих на `finance.yahoo.com`.

Функция Альтера-Джонса

В задачах анализа временных рядов нередко возникает ситуация, при которой необходимо из данных извлечь колебательную составляющую для того, чтобы оставить только трендовые движения показателя. Данная задача успешно решается многими методами очистки данных от шума (сглаживание), поиска периодической зависимости в данных (спектральное преобразование Фурье, автокорреляционная функция).

В данной работе для реализации в виде функции предлагается рассмотреть специальную функцию для поиска периодических компонент временного ряда – функцию *Альтера-Джонса*:

$$a_{\tau} = \frac{1}{n-\tau} * \sum_{i=1}^{n-\tau} |y_{i+\tau} - y_i|, \quad (12)$$

где n – длина исходного ряда, τ – область определения функции или смещения для входного ряда y , y – входной временной ряд для которого исследуется периодичность.

Функция Альтера-Джонса по своей природе показывает минимум в точке τ^* – смещения, определяемого как «почти-период». Собственно данное значение является характерным временем повторения очередного цикла похожих событий на временном ряду.

Периодичность данных, в которых присутствует трендовая компонента по факту сложно исследуема, поэтому для того, чтобы в исходных данных искать периодическую зависимость необходимо предварительно подготовить y изъав трендовую зависимость из исходных данных, сохраняя качественные особенности колебаний.

Данную задачу в рамках самостоятельной работы предлагается решить с помощью следующих функций:

средняя арифметическая пропорция:

$$y_t = \ln \left(\frac{x_{t-dt} + x_{t+dt}}{2x_t} \right), \quad (13)$$

средняя геометрическая пропорция:

$$y_t = \ln \left(\frac{x_{t-dt} * x_{t+dt}}{x_t^2} \right), \quad (14)$$

средняя гармоническая пропорция:

$$y_t = \ln \left(\frac{2 * x_{t-dt} * x_{t+dt}}{x_t * (x_{t-dt} + x_{t+dt})} \right), \quad (15)$$

где x – исходный вектор временного ряда с трендом, t – индекс массива или вектора, dt – пробный сдвиг по индексам (или времени), определяющий степень извлечения информации из данных. Чем больше показатель dt тем меньше будет элементов в векторе y , и тем более явно будут выражены большие колебания в исходных данных.

Данные функции изъятия тренда стоит применять перед использованием функции Альтера-Джонса только в случае наличия в данных устойчивой трендовой составляющей.

Задание 1.

1. Внимательно изучить скрипт, загружающий финансовые данные в среду R. Выполнить данный скрипт в вашем проекте самостоятельной работы

2. Реализовать функцию исключения тренда:

`out_of_trend(x, dt, method = c("Arifm", "Geom", "Garm"))`,

которая принимает на вход ряд чисел x , пробные смещения dt , метод решения данной задачи `method`. Функция на выходе должна возвращать также ряд чисел, величина которого меньше на $2 * dt$ (подумайте почему), который получился в результате применения одного из трёх методов.

Все методы в функции должны быть реализованы, выбор метода осуществляется только при вызове функции. В стандартном случае или в любом другом случае вызывать среднюю арифметическую пропорцию.

Исключительными случаями для данного метода являются: ввод вектора длины меньше 3, ввод dt больше чем $\text{ceiling}(x/2)-1$, ввод нечислового вектора, ввод нечислового пробного смещения. Обработать при реализации эти случаи.

Перед применением функции к ряду с трендом поднять исходный ряд по оси ординат на величину самого минимального значения вектора плюс единица. Это нужно для корректной работы натурального логарифма.

3. Задать векторы $t = \text{seq}(0, 10, 0.1)$ и $x = 2 * t + 3 + \sin(2 * t)$.

Подсчитать среднее от вектора x . Применить функцию извлечения тренда к вектору x и записать его в переменную xn . Подсчитать среднее xn . Дать ответ, почему среднее xn находится около нуля.

4. Реализовать функцию Альтера-Джонса

`Alter-Johns(y)`,

где y – временной ряд (вектор чисел) без тренда. Подумайте, по определению функции Альтера-Джонса, каким является область определения для неё, т.е. в каком диапазоне могут лежать значения t .

5. Применить функцию Альтера-Джонса к вектору `xp`. Какое `t` отвечает *локальному* минимуму функции Альтера-Джонса?

6. Применить разработанные функции к одной из колонок таблицы данных `df`, загруженных ранее, и сохранить результат работы функции Альтера-Джонса по данному ряду с исключенным трендом в отдельном векторе. Подставьте данный вектор в функцию `plot()` и понаблюдайте за её поведением. В какой точке наблюдается первый локальный минимум.

Внимание. Методы из пунктов 2 и 4 могут быть реализованы в отдельных файлах `*.R`, перед тестированием они должны быть последовательно вызваны единожды для того, чтобы они сохранились в окружении глобальных функций.

Часть 3

Итерационный метод

Данное задание снова относится к реализации метода простой итерации из двух прошлых самостоятельных работ. В данной работе будет необходимо реализовать функцию, обеспечивающую решение СЛАУ методом простой итерации, и обработать все её исключительные случаи в процессе реализации. Функция, реализующая метод простой итерации должна выглядеть следующим образом:

`SIM(A, u0, f, n_iter = 10e5, eps = 10e-7)`

Исключительные случаи метода были разобраны ранее в работе №5.

Задание 1.

Реализовать функцию, осуществляющую решение СЛАУ методом простой итерации, соблюдая поставленные ранее условия.

Контрольные вопросы

1. Что понимается под функцией в программировании. В чем особенности функций в R
2. Какие параметры используются в функциях
3. В чем отличие позиционных параметров от ключевых параметров
4. Чем отличаются формальные и фактические параметры. Где они указываются
5. Каким образом могут быть заданы параметры функции
6. Каким образом функция в R может вернуть несколько значений
7. Как оформляется тело функции

Практическая работа №7

Двухмерная графика

В R можно строить как плоские графики, так и объемные. Вначале рассмотрим первый случай. Приведем описание полезных, на наш взгляд, функций, раскрывая только их основные параметры.

Функция `plot()` предназначена для первоначального построения графиков функций на плоскости:

```
plot(x, f, type, fty, xlim, ylim, main, sub, xlab, ylab, col, lwd)
```

Первый позиционный параметр `x` представляет собой вектор – множество точек на оси абсцисс в указанном диапазоне, для которых будут вычислены значения функции и на их основе будет построен график.

Второй позиционный параметр `f` представляет собой функцию, с помощью которой будут вычислены точки графика, соответствующие значениям точек вектора `x`.

Остальные параметры функции – ключевые; в большинстве своем они являются строковыми.

Параметр `type` определяет вид графика: "l" – линии, "p" – точки, "b" – кружочки, "o" – линии с кружочками, "h" – вертикальные линии (гистограммы), "s" – ступеньки,...

Параметр `lty` задает вид линии: "solid" – сплошная, "dashed" – пунктирная, "dotted" – точками,

Параметры `xlim`, `ylim` устанавливают границы графика по осям `Ox` и `Oy` (соответственно).

Параметры `main`, `sub`, `xlab`, `ylab` предписывают вывести на график надписи – основной заголовок, подзаголовок, подписи осей `Ox` и `Oy` (соответственно).

Параметр `col` определяет цвет линий графика; "black" – черный, "red" – красный, "green" – зеленый, "blue" – синий, "gray" – серый,...

Параметр `lwd` – число задает толщину линий графика (1 – стандартная толщина, 2 и более – увеличенная толщина).

Функция `lines()` предназначена для добавления на построенный график с помощью функции `plot()` новых графиков: `lines(x, f, type, lty, col, lwd)`

Назначение параметров данной функции – `x`, `f`, `type`, `lty`, `col`, `lwd` аналогично параметрам функции `plot()`.

Функция `abline()` позволяет вывести на построенный график дополнительные элементы:

```
abline(a, b, h, v)
```

Числовые значения параметров a и b указывают на вывод линии, заданной уравнением $y = a + bx$.

Значение параметра $h=y$ предписывает вывести горизонтальную линию $y=h$.

Значение параметра $v=x$ предписывает вывести вертикальную линию $x=v$.

Функция `text()` добавляет к графику текст:

`text(x, y, labels, col)`

Параметры x и y определяют координаты, куда будет добавлен текст, заданный в параметре `labels`.

Параметр `col` задает цвет символов текста.

Функция `points()` добавляет точки к графику:

`points(x, y, type, col, pch, bg)`

Параметры x и y определяют координаты добавляемой точки,

Параметр `col` задает цвет точки.

Параметр `pch` задает вид точки (указывается в виде целого числа).

Параметр `bg` определяет цвет внутреннего заполнения точки.

Приведем несколько примеров для демонстрации возможностей указанных функций.

Пример.

Объявить пользовательскую функцию $g = c|x|^{0.5}$

Используя данную функцию, построить с помощью функций `plot()`, `lines()`, `abline()` ряд графиков с различными значениями параметров исследуемой пользовательской функции. Добавить на график с помощью функций `text()` и `points()` оформительские элементы.

Решение

Протокол скрипта

```

2  # построение двумерного графика
3  g=function(x,c=1) c*abs(x)^0.5
4  x=seq(-5,5,length=101)
5  x
6  plot(x,g(x),
7       type="l",
8       lty="solid",
9       main="Графики функции g(x)",
10      sub="25 января 2021 г.",
11      xlab="ось x",
12      ylab="ось y",
13      col="blue",
14      lwd=1)
15  abline(h=0,v=0,col="gray50",lty="solid")
16  lines(x,g(x,c=0.5),type="h",
17        lty="dashed",col="red")
18  lines(x,g(x,c=2),lty="dotted",col="blue")
19  abline(a=1,b=0.5,col="green",lwd=2)
20  text(3.4,2.2,"y=0.5x+1",col="black")
21  text(-3,2.2,"Точка",col="red")
22  points(-3,2,col="black",pch=11,bg="yellow")

```

Результат построения графика показан на рисунке 45.

Графики функции g(x)

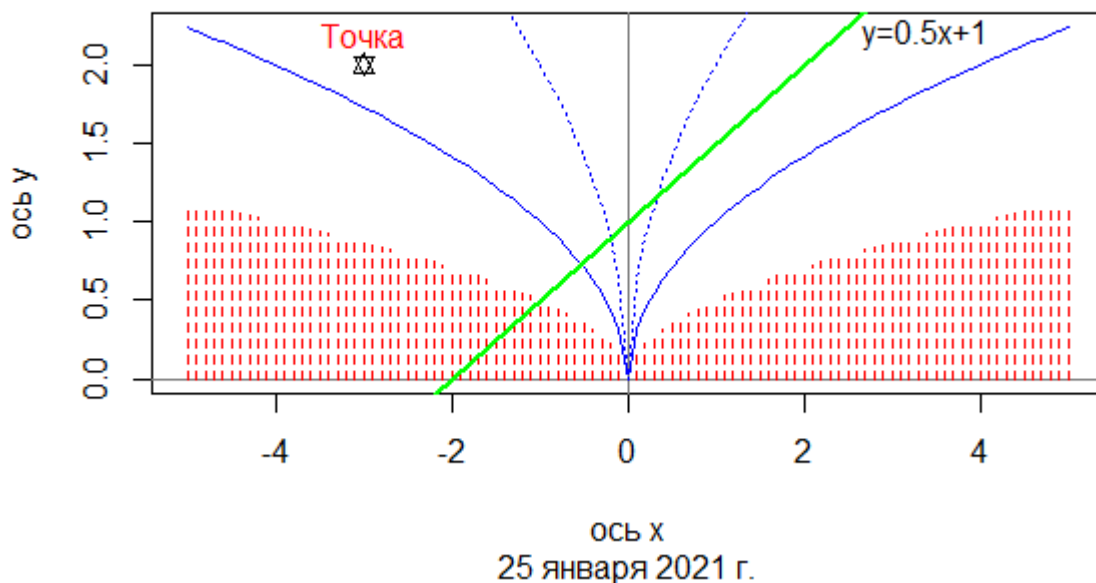


Рисунок 49 Пример построения двумерного графика

Трёхмерные графики и линии уровня

Трёхмерные, объёмные графики предоставляют более широкие возможности для анализа и исследования поведения функций.

Для построения таких графиков используется функция persp():

```

persp(x = seq(0,1, length.out = nrow(z)),
      y = seq(0,1, length.out = ncol(z)),
      z,
      xlim = range(x), ylim = range(y),

```

```
zlim = range(z, na.rm = TRUE),  
xlab = NULL, ylab = NULL, zlab = NULL,  
main = NULL, sub = NULL,  
theta = 0, phi = 15, r = sqrt(3), d = 1,  
scale = TRUE, expand = 1,  
col = "white", border = NULL, ltheta = -135, lphi = 0,  
shade = NA, box = TRUE, axes = TRUE, nticks = 5,  
ticktype = "simple",...)
```

Параметры *x* и *y* представляют собой упорядоченные по возрастанию координаты линий сетки, для которых специальным образом вычисляются с помощью функции `outer()` значения матрицы *z* для любых комбинаций *x* и *y*.

Параметры *theta* и *phi* задают углы обзора объемной фигуры по горизонтали и вертикали.

Параметр *col* определяет цвет графика (дополнительная информация в справке по функции `colors()`).

Параметр *ticktype* устанавливает маркировку осей: "detailed" – точные значения, "simple" – только стрелки.

Параметры *xlim*, *ylim*, *main*, *xlab*, *ylab* аналогичны соответствующим параметрам функции `plot()`.

Функция `outer()` позволяет рассчитать для исследуемой функции *f* значения элементов матрицы для любых комбинаций векторов *x* и *y*:

```
outer(x, y, f)
```

При исследовании объемных фигур, отображающих поведение многомерных функций, полезным является использование изолиний – линий уровня на графике, в каждой точке которых скалярный результат зависимости от двух переменных на плоскости сохраняет одинаковое значение.

Поскольку изолинии наглядно отображают характер изменения величин пространственно-непрерывных явлений, они, кроме математики, широко используются и в других областях деятельности. Так, при изображении физических и топографических карт местности изолинии характеризуют рельеф указанием одинаковых значений высоты над уровнем моря.

В метеорологии используют изотермы и изобары – соответственно изолинии одинаковых температур и одинакового давления. В экономике применяется термин изокванты – изолинии одинакового объема производства продукта в зависимости от факторов производства.

Для построения изолиний объемных фигур используется функция `contour()`:


```

contour(x = seq(0,1, length.out = nrow(z)),
        y = seq(0,1, length.out = ncol(z)),
        z,
        nlevels = 10, levels = pretty(zlim, nlevels),
        labels = NULL,
        xlim = range(x, finite = TRUE),
        ylim = range(y, finite = TRUE),
        zlim = range(z, finite = TRUE),
        labcex = 0.6, drawlabels = TRUE, method = "flattest",
        vfont, axes = TRUE, frame, plot = axes,
        col = par("fg"), lty = par("lty"), lwd = par("lwd"),
        add = FALSE,...}

```

Большинство параметров функции имеют значение, аналогичное параметрам функции `persp()`.

Параметр `nlevels` определяет количество линий уровня.

Параметр `vfont` позволяет определить семейство шрифтов для отображения меток изолиний.

Параметр `label` задает вектор определения «высот» в изолиниях, отличных от стандартных значений.

Демонстрацию возможностей рассмотренных функций по построению объемных графиков выполним на примерах, в которых предварительно объявим пользовательские функции, затем построим их графическое отображение при определенных значениях параметров, используя функцию `persp()`, `outer()`, `contour()` и др.

Пример

Объявить пользовательскую функцию $f = c + 2x^\alpha - 4y^\alpha$.

С помощью функции `persp()` и с привлечением других функций R построить объемный график функции `f` и вывести ее изолинии.

Выполнить исследование полученных результатов при различных значениях входных параметрах.

Решение

Текст скрипта:

```

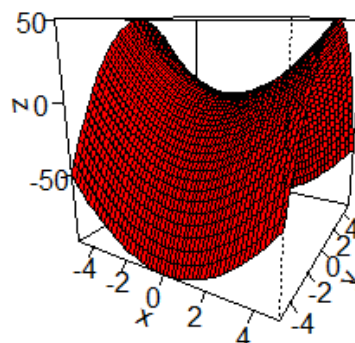
1 # Построение трехмерного графика
2 f<-function(x=0.5,y=0.5,c=1,a1=2) c+2*x^a1-4*y^a1
3
4 x<-y<-seq(-5,5,length=51)
5 x
6 y
7 z<-outer(x,y,f)
8 persp(x,y,z,
9       theta = 30,phi=15,
10      col="red",
11      ticktype="detailed",
12      main="Графики функции",
13      sub="f=c+2x^a1-4y^a1",
14      )
15 contour(x,y,z,
16         nlevels=10,
17         col="black",
18         main="Изолинии функции f",
19         sub="f=c+2x^a1-4y^a1")

```

В результате выполнения введенных операторов (рекомендуется их выполнить по одному, чтобы уяснить действие каждого из них) получим вычисленные значения векторов x и y , а также матрицы z , значения которых служат исходными данными для построения графиков с помощью функций `persp()` и `contour()`.

Трехмерный график функции представлен на рисунке 47.

Графики функции



$$f=c+2x^{a1}-4y^{a1}$$

Рисунок 50 Пример трехмерного графика функции

Примечания

1. В тексте скрипта вывод значений векторов x и y , а также матрицы z на экран указан в качестве комментария, так как эти объекты содержат большое количество элементов, и протокол вывода будет очень массивным. Рекомендуется при выполнении скрипта обеспечить вывод объектов x , y и z на экран.

2. Для просмотра созданных в скрипте графиков и вызванных справок следует выполнить листание на вкладках окна RStudio Plots и Help (соответственно).

Контурное изображение графика представлено на рисунке 48.

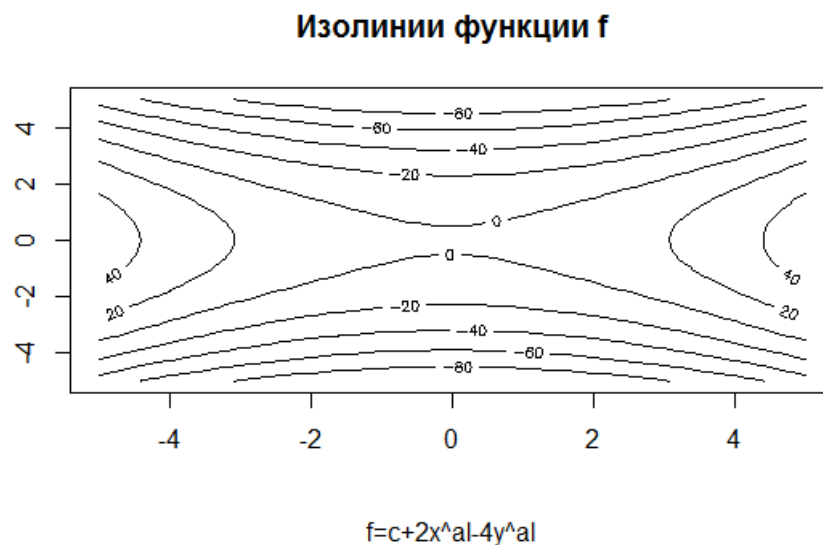


Рисунок 51 Контурное изображение графика

Пример.

Текст скрипта

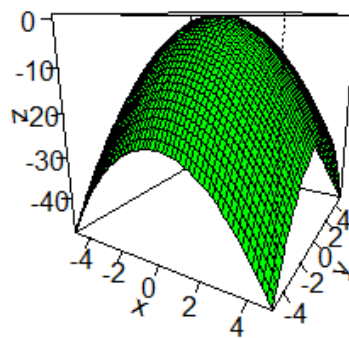
```

2 # построение графика и изолиний трехмерной фигуры
3 f<-function(x1,x2,c=1,a=2) c-x1^a1-x2^a1
4
5 x<-y<-seq(-5,5,length=51)
6 x
7 y
8 z<-outer(x,y,f)
9 persp(x,y,z,
10       theta = 30,phi=10,
11       col="green",
12       ticktype="detailed",
13       main="Графики функции",
14       sub="f=c-x^a1-y^a1",
15 )
16 # построение изолиний трехмерной фигуры
17 contour(x,y,z,
18         nlevels=5,
19         col="black",
20         main="Изолинии функции f",
21         sub="f=c+2x^a1-4y^a1")

```

Трехмерный график функции представлен на рисунке 49.

Графики функции

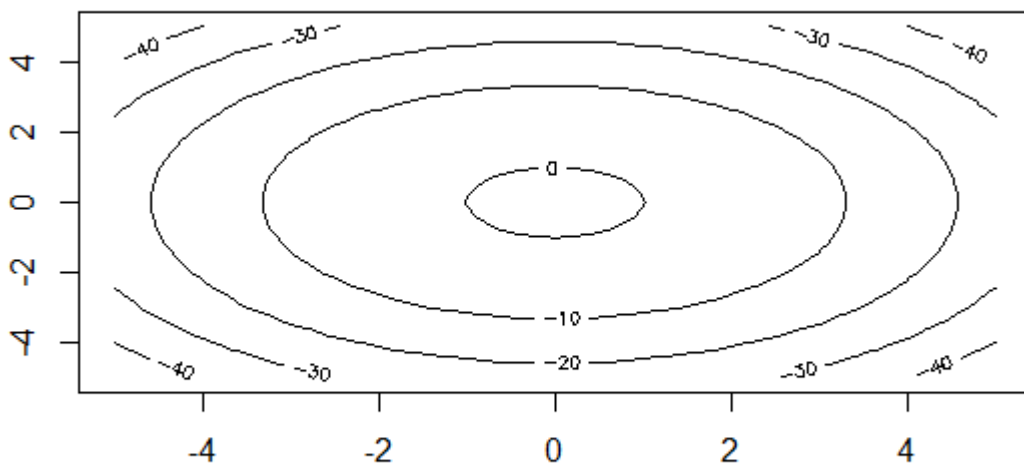


$$f = c - |x|^a - |y|^a$$

Рисунок 52 Пример трехмерного графика

Изолинии графика представлены на рисунке 50.

Изолинии функции f



$$f = c - |x|^a - |y|^a$$

Рисунок 53 Пример линий уровня функции $f = 1 - x^2 - y^2$

Самостоятельная работа

1. Разработать программный скрипт вычисления значений функции $y = x^2$. Построить график. Нанести на график текущую дату, предварительно самостоятельно изучив возможности работы с датами.
2. Разработать программный скрипт ввода данных об успеваемости 4-х студентов группы по трем изучаемым дисциплинам. Построить круговую диаграмму успеваемости третьего студента группы.

Оценить изменения программного скрипта для случаев, когда число студентов варьируется до 25 человек, а количество предметов – до 5.

3. Выполнить комплексное исследование гиперболической функции:

$$f(x) = \frac{3x^2 - 30x + 74,417}{x - 5} \quad (16)$$

Найти нули функции (корни) и ее экстремальные точки (нули производной). Построить графики функций.

4. Выполнить исследование кубической функции:

$$f(x) = x^3 - 18x^2 + 106,25x - 205,5 \quad (17)$$

Найти нули функции (корни) и ее экстремальные точки (нули производной). Построить графики функции и ее производной.

5. Разработать пользовательскую функцию и построить трехмерное изображение эллиптического цилиндра.

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \quad (18)$$

6. Разработать пользовательскую функцию и построить трехмерное изображение гиперболического цилиндра.

$$\frac{x^2}{a^2} - \frac{y^2}{b^2} = 1 \quad (19)$$

7. Разработать пользовательскую функцию и построить трехмерное изображение однополостного гиперболоида.

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} - \frac{z^2}{c^2} = 1 \quad (20)$$

8. Разработать пользовательскую функцию и построить трехмерное изображение тора.

$$(x^2 + y^2 + z^2 + R^2 - r^2) - 4R^2(x^2 + y^2) = 0 \quad (21)$$

Контрольные вопросы:

1. Какие функции используются для создания плоских графиков
2. В чем схожесть и в чем отличие функций `plot()` и `lines()`
3. Какие функции используются для построения трехмерных графиков
4. Каковы возможности по анализу трехмерных фигур
5. Каким образом можно построить изолинии трехмерных фигур

Практическая работа №8

Некоторые сведения о возможностях статистического анализа

Затабулированные распределения

Пакет R позволяет проводить различные виды статистической обработки данных, информация о некоторых из них приведена в таблице.

Таблица 6

Затабулированные распределения в R

| Название | Обозначение в R | Параметры |
|-----------------|-----------------|--------------|
| Нормальное | norm | mean, sd |
| t-распределение | t | df |
| Равномерное | unif | min, max |
| Хи-квадрат | chisq | df |
| F-распределение | f | df1, df2 |
| Гамма | gamma | shape, scale |

Стандартное нормальное распределение

Многие естественные процессы соответствуют закону нормального распределения. В R есть группа взаимосвязанных функций для работы с такими данными.

Функция rnorm()

Служит для генерации совокупности нормально распределенных случайных чисел:

```
rnorm(n, mean = 0, sd = 1)
```

Параметр n представляет собой объем создаваемой случайной выборки.

Параметр mean – математическое ожидание.

Параметр sd – среднее квадратическое отклонение.

Функция qnorm()

Предназначена для вычисления квантилей нормального распределения:

```
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
```

Параметр p представляет значение вероятности.

Параметры mean и sd аналогичны параметрам функции rnorm().

Параметр lower.tail – логическая величина: для значения TRUE (по умолчанию) вероятность рассчитывается как $P[X < x]$, иначе – как $P[X > x]$.

Параметр `log.p` – логическая величина; если его значение `TRUE`, вероятности `p` задаются как `log(p)`.

Функция `pnorm()`

Является функцией распределения и описывает вероятность того, что случайная переменная X примет какое-либо значение, не превышающее либо равное x :

`Pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)`

Параметр `q` – квантиль случайной величины.

Параметры `mean`, `sd`, `lower.tail`, `log.p` соответствуют аналогичным параметрам функций `gnorm()` и `qnorm()`.

Функция `dnorm()`

Рассчитывает значения функции плотности вероятности для заданных значений вектора x :

`Dnorm(x, mean = 0, sd = 1, log = FALSE)`

Параметр x – вектор значений случайной величины. Параметр `log` – логическая величина; если его значение `TRUE`, вероятности `p` задаются как `log(p)`.

Иллюстрацию возможностей указанных функций приведем в примере.

Пример.

С помощью функций `gnorm()`, `qnorm()`, `pnorm()` и `dnorm()` сформировать массив случайных чисел.

```
2 n<-rnorm(100,mean = 15,sd=5)
3 n
4 hist(n)
```

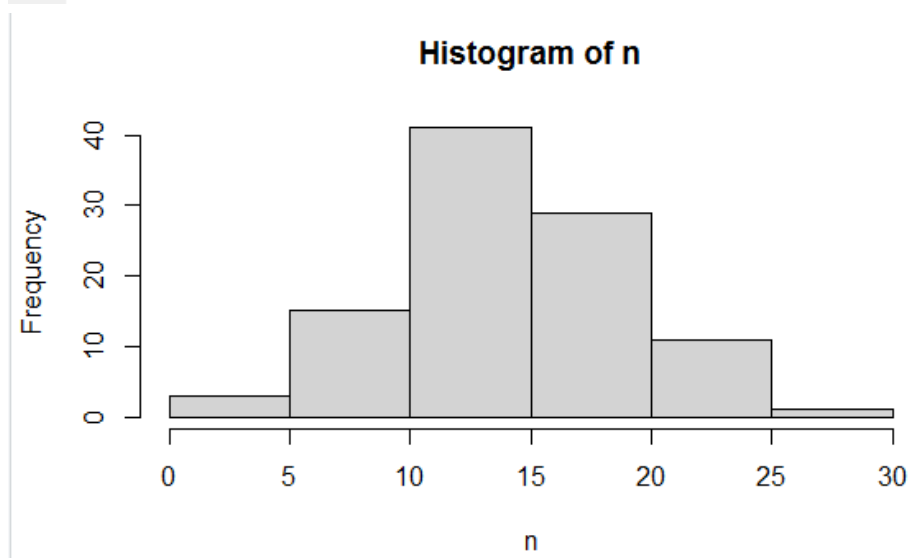


Рисунок 54 Графическое представление выборки

Функция `qnorm()` принимает значение вероятности и дает число, совокупное значение которого соответствует значению вероятности.

Например, предположим, что мы хотим найти 85-й процентиль нормального распределения, среднее значение которого равно 70 и стандартное отклонение которого равно 3.

```
6 x <- seq(0, 1, by = 0.02)
7 y <- qnorm(x, mean = 2, sd = 1)
8 plot(x,y)
```

График функции распределения представлен на рисунке 52.

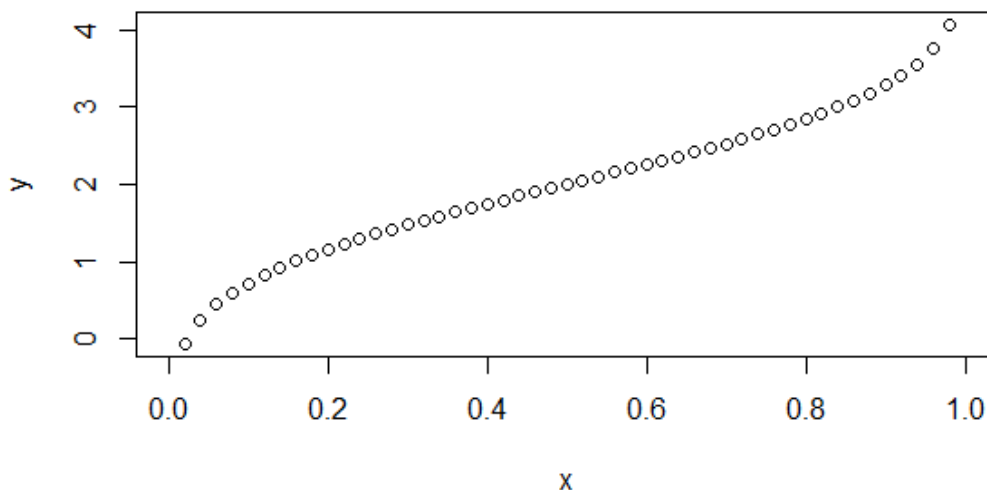


Рисунок 55 График функции распределения

Функция `pnorm()` дает вероятность того, что обычно распределенное случайное число меньше значения данного числа. Она также называется «кумулятивная функция распределения».

```
8 x <- seq(-10,10,by = .2)
9 x
10 y <- pnorm(x, mean = 2.5, sd = 2)
11 y
12 plot(x,y)
```

График функции распределения представлен на рисунке 53.

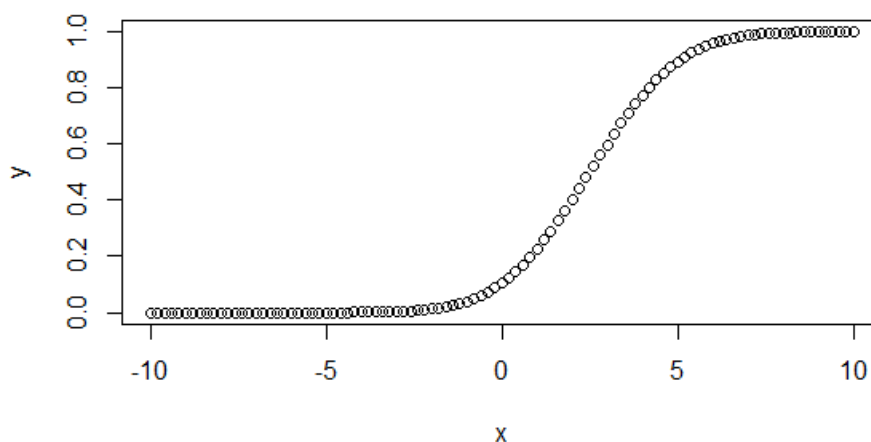


Рисунок 56 Графическая иллюстрация функции распределения нормально распределенных чисел

Функция `dnorm()` дает высоту распределения вероятностей в каждой точке для данного среднего значения и стандартного отклонения.

```
16 x <- seq(-10, 10, by = .1)
17 y <- dnorm(x, mean = 2.5, sd = 0.5)
18 plot(x,y)
```

Графическая иллюстрация функции плотности нормально распределенных чисел представлена на рисунке 54.

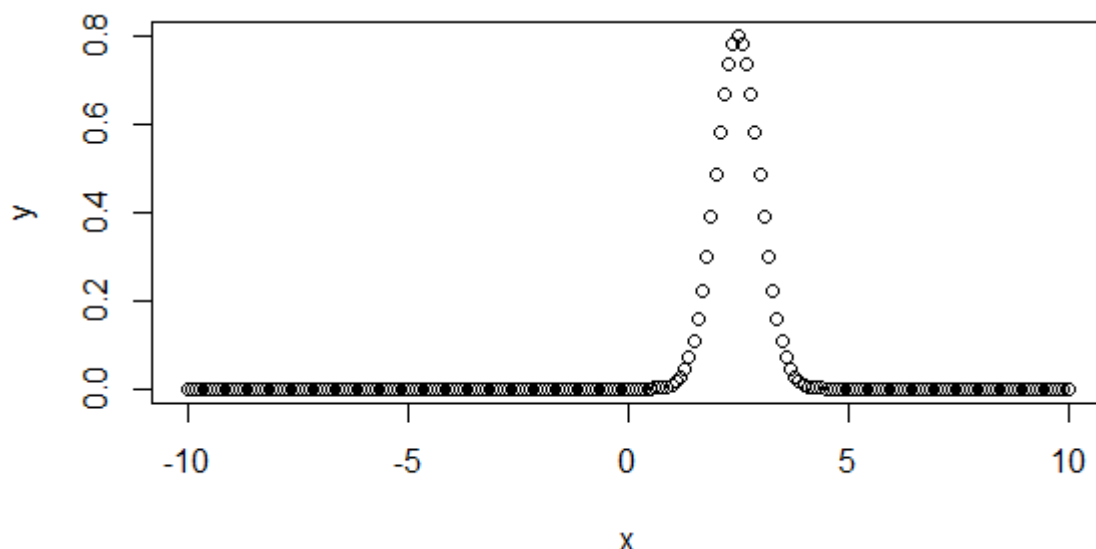


Рисунок 57 Графическая иллюстрация функции плотности нормально распределенных чисел

Гистограмма и эмпирическая плотность

При работе с пакетом R исследователю предоставляется возможность доступа к многочисленным массивам данных. Задав необходимые дополнительные значения, можно исследовать, например, понятие эмпирической плотности, нанесенной на гистограмму.

Не вдаваясь в теоретические подробности, приведем пример скрипта с подробными комментариями, решающего указанную задачу, и результаты его выполнения.

```
1 # Эмпирическая плотность
2 y<-faithful$eruptions # исходные данные
3 y
4 hist(y,breaks=seq(1.6,5.2,by=0.2),prob=T,
5      main="Гистограмма",
6      ylab="Плотность",
7      xlab="Диапазон данных")
8 # Расчет значений эмпирической плотности
9 y.pdf<-density(y,bw="ucv")
10 # Вывод кривой эмпирической плотности
11 lines(y.pdf,col="red")
12 text(3.1,0.5,"Эмпирическая плотность")
13 # добавление исходных данных на ось OX:
14 rug(y)
```

Рисунок 58 Текст скрипта построения гистограммы с нанесенной эм-

пирической плотностью

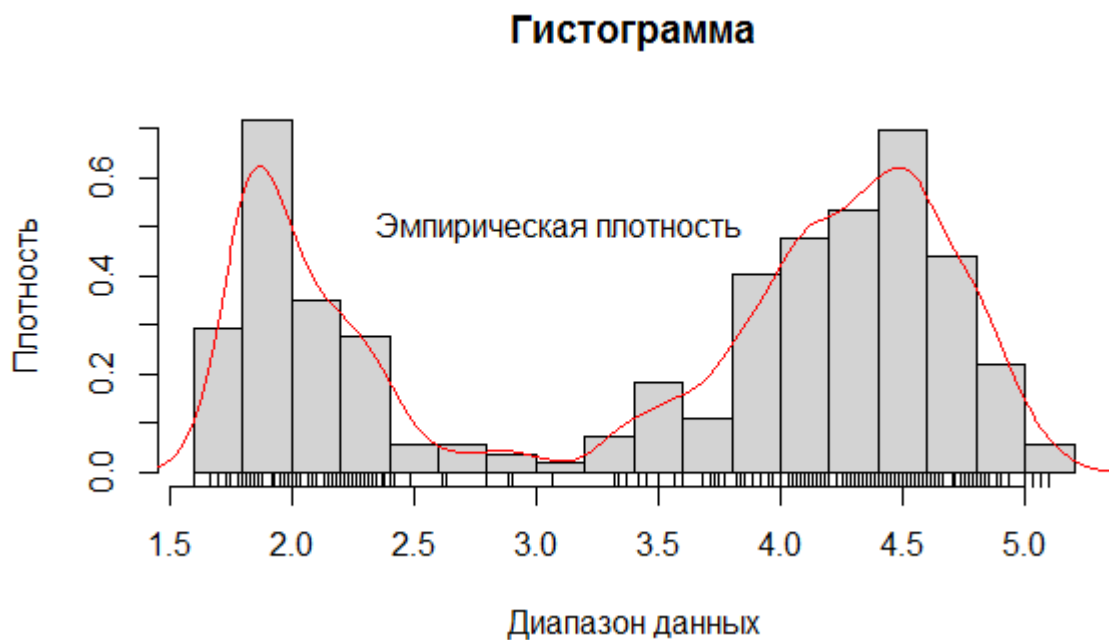


Рисунок 59 Гистограмма с нанесенной эмпирической плотностью

Эмпирическая функция распределения

Эмпирической функцией распределения (функцией распределения выборки) называется функция $F^*(x)$, определяющую для каждого значения x частоту события $X < x$.

Приведем текст скрипта, демонстрирующего возможность исследования эмпирических функций распределения, а также графический результат его выполнения.

Текст скрипта:

```
1 # Эмпирическая плотность
2 y<-faithful$eruptions # исходные данные
3 y.cdf<-ecdf(y)
4 # y.cdf - функция, подставляя в которую квантили,
5 # дает значения эмпирической функции распределения
6 y.cdf(3) # значение функции распределения для квантили 3
7
8 plot(y.cdf,do.points=F,verticals=T,
9      main="График эмпирической функции распределения",
10     ylab="Значения функции распределения",
11     xlab="диапазон данных")
12 points(3,y.cdf(3),col="red",pch=10)
13 # вывод на график значения эмпирической функции
14 # распределения для квантиля 3
15 text(3,0.5,y.cdf(3))
```

Результат выполнения скрипта представлен на рисунке 56.

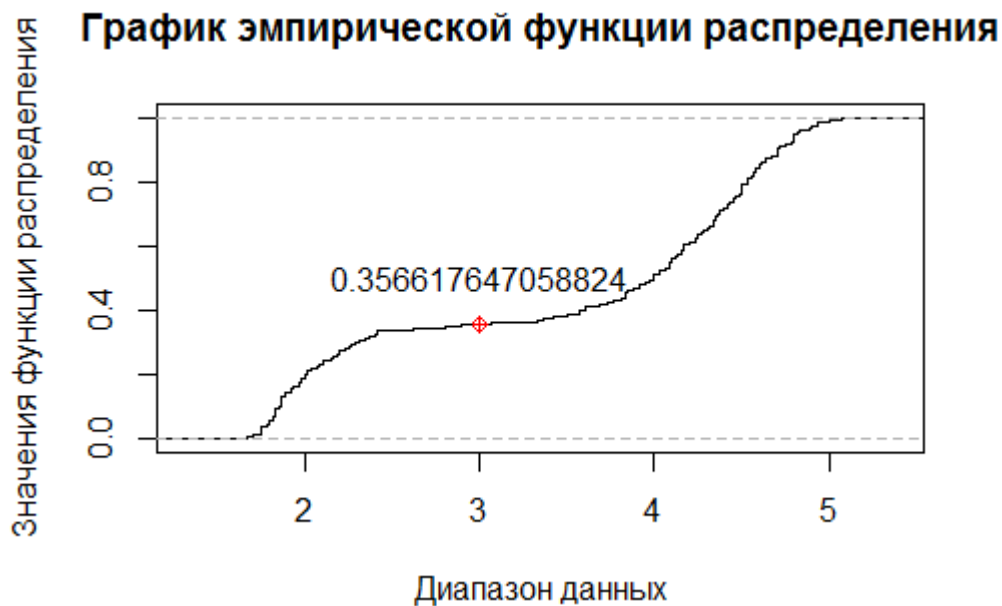


Рисунок 60 Результат выполнения скрипта исследования эмпирической функции распределения

Самостоятельная работа №8

Исследование статических и вероятностных характеристик временных рядов

В работе рассмотрены данные временного ряда, полученные в результате эксперимента снятия физиологических показателей, связанных с электрической активностью организма в покое и при различных внешних психических воздействиях. Вес подопытного в момент эксперимента – 90 кг, рост подопытного в момент эксперимента – 195см.

Данные можно получить по следующей ссылке:
https://raw.githubusercontent.com/qwerty29544/RpracticeBook/master/2Data/01FlatTables/ECG_yurchenkov.txt.

Работа является больше исследовательской с точки зрения отработки навыков предобработки данных в R, анализа статистических закономерностей в данных, графического анализа.

Функциональное требование: вся обработка текстового файла должна производиться с помощью средств R. Необходимо, чтобы все переменные анализа:

- частота дискретизации,
- единицы измерения,
- названия характеристик,
- сами данные,

были внесены в программный скрипт только из файла с данными и только при помощи R.

Задание 1.

1. Скачать данные по указанной ссылке при помощи функции `download.file()`.

2. Произвести импорт данных в среду R различными методами функционалом языка R.

3. Все метаданные таблицы измерений обработать при помощи R различными способами, результатом должны являться переменные, содержащие метаданные каналов снятия измерений. Если для обработки данных понадобятся метаданные по росту и весу, разрешается внести эти данные как переменные среды внутри скрипта.

4. Функционально установить количество стадий эксперимента, т.е. сколько опытов в процессе эксперимента проводилось над подопытным.

5. Установить точное время в мс точек снятия показателей в связи с полученной частотой дискретизации каналов снятия измерений и номерами строк. Считать время с 0минут:00секунд:00миллисекунд.

6. Построить гистограммы распределения каждого из временных рядов в целом и по выявленным стадиям эксперимента. Предоставить сводку по описательным статистикам данных как полностью, так и по стадиям эксперимента.

7. Предложить свои варианты методов обработки данных исходя из ранее полученных навыков и разработанных функций в рамках выполнения самостоятельной работы №6.

Важно. Для выполнения такого рода задания вам, возможно, понадобится знание организации численного вектора гистограммы, т.е. независимого получения количества элементов в столбцах данных:

```
table(cut(x = ts, breaks = seq(min_ts, max_ts, (max_ts - min_ts)/n))),
```

где `ts` – временной ряд или просто числовой вектор данных, `min_ts` и `max_ts` – естественные границы данных по оси ординат, т.е. естественный максимум и минимум данных, `n` – количество столбцов данных.

Функционально разбиения можно организовать и другими способами, важным является критерий – это должен быть вектор чисел.

Контрольные вопросы:

1. Приведите примеры функций обработки стандартных нормальных распределений.
2. Каким образом могут быть построены гистограммы в R

Практическая работа №9

Тестирование в testthat

При разработке функционала обработки данных в R не так уж и часто необходимо проверять правильность работы своего программного кода ввиду того, что зачастую программы на данном языке маленькие и выполняют очень узкую специализированную задачу. В анализе данных на R так и вообще используется очень много готового функционала в виде сторонних пакетов, функции в которых уже протестированы и степень их покрытия тестами очень велика.

Однако бывают случаи, когда при разработке программного кода на R всё же необходимо произвести тестирование разработанных классов или функций. Делать это очень удобно с помощью пакета testthat. Данный пакет включён в список обязательных пакетов для разработки комплексного программного обеспечения на R и требуется при запуске компиляции законченных проектов и решений.

При загрузке пакета testthat и подключению его к среде выполнения программы в окружении testthat:: будет доступно множество функций для организации блочного тестирования, а также тестирования «на ходу», т.е. специальных проверок, работающих по индивидуальной логике. Важной особенностью тестов является немедленная остановка выполнения кода скрипта в случае, если тест является проваленным. Базовый список функций для тестирования:

```
# Функция, настраивающая автоматическое тестирование скрипта
testthat::auto_test(code_path, test_path)
```

```
# Функция, позволяющая формировать блоки тестов
testthat::test_that("Описание действия теста", {
  # Код теста
})
```

```
# Набор функций, начинающихся с expect_, проверяют возврат какой-либо
разработанной функции
testthat::expect_
```

Пример

```
# Не возвращает ничего – тест пройден
testthat::expect_equal(object = sum(c(5, 6)), expected = 5 + 6)

# Возвращает ошибку, тест не пройден
testthat::expect_equal(object = sum(c(5, 6)), expected = 5 + 7)
```

Ошибка: `sum(c(5, 6))` not equal to `5 + 7`.

1/1 mismatches

[1] `11 - 12 == -1`

Таким образом, при помощи данных assert-ов и автоматизации тестирования, вынесения тестов в отдельный файл можно достичь продуктивности разработки функции при проработке их исключительных или граничных ситуаций.

Самостоятельная работа №9

Тестирование функций

Задание 1.

1. Внимательно изучить возможности команд семейства `testthat::expect_`

2. Настроить файлы автоматического тестирования функций метода простой итерации, функции извлечения тренда из данных, функции Альтера-Джонса. Для выполнения данного задания, необходимо реализованные ранее в самостоятельной работе №6 функции переоформить в качестве отдельных файлов скрипта, создать отдельные файлы с тестами под каждую из созданных функций и при помощи:

```
testthat::auto_test(code_path, test_path)
```

настроить выполнение автоматических тестов.

3. Разработать в виде блоков тестов функциональные проверки граничных и исключительных случаев для функций, рассмотренных ранее в самостоятельной работе №6. Для выполнения данного пункта необходимо использовать блочные тесты

```
# Функция, позволяющая формировать блоки тестов
testthat::test_that("Описание действия теста", {
  # Код теста
})
```

Необходимо описывать в строке суть теста и суть внутренних проверок.

4. Минимальный объем проверок – 6 тестов на каждую функцию. Важно понимать, что больше – лучше.

Проверка выполнения задания осуществляется через сохранение файла и демонстрации возможного падения теста в результате изменения кода функций и обратный возврат к прохождению тестов путём возврата к предыдущей версии до изменений.

Практическая работа №10

Построение графики в пакете ggplot2

Построение графики с пакетом ggplot2 не очень сильно отличается от использования стандартных графических возможностей R, однако степень возможной вариации и детализации результата действительно значительно выше по сравнению с классическим вариантом графики из пакета graphics::.

Установка демонстрационного скрипта:

```
# Скачивание ggplot2 в случае необходимости
if ("ggplot2" %in% rownames(installed.packages()) == FALSE) {
  install.packages("ggplot2") }
library(ggplot2)

# Существующий набор данных
df <- faithful
```

Графика в ggplot2 оперирует понятиями “холст” и “слой”. Холст в ggplot2 определяется указанием функции ggplot() в теле скрипта. При выполнении данной функции на выходе мы получаем открытый графический девайс R и пустой лист, на котором ничего не имеется.

Для определения того, что нужно отрисовать на графике необходимо указать в качестве аргументов data — фрейм данных, по которым будет строиться график. В аргументе mapping мы должны указать функцию aes(), в которой мы определим оси и размерность измерений.

Пример определения данных для графика в ggplot2:

```
ggplot(data = df, mapping = aes(eruptions, waiting))
```

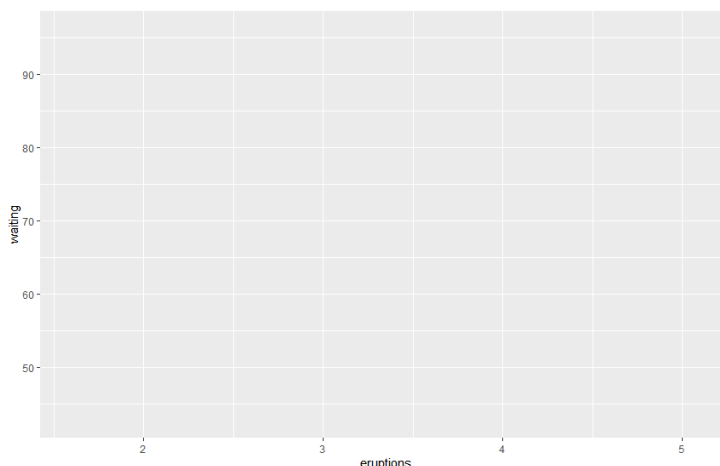


Рисунок 61 Пустой холст графика. Видны оси, видна сетка определения осей

Для того, чтобы на данный холст добавить маркеров или другой графики необходимо указать требуемый слой для построения из предиктивы `geom_`. Данные функции, начинающиеся с `geom_` определяют слои, которые мы хотим нанести на график:

```
ggplot(data = df, mapping = aes(eruptions, waiting)) +  
  geom_point()
```

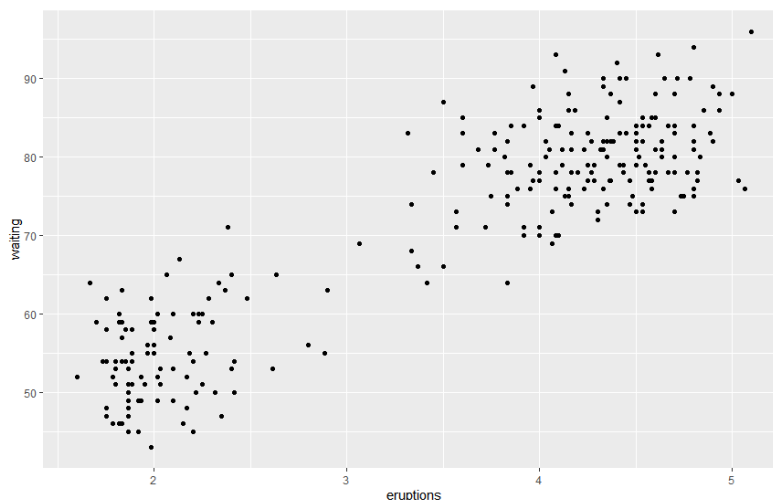


Рисунок 62 На холст графика нанесён слой точек

Слои и холсты в `ggplot2` можно очень долго изменять, добиваясь безупречных результатов графического отображения:

```
ggplot(data = df, mapping = aes(eruptions, waiting)) +  
  geom_point(color = kmeans(x = df, centers = 2)$cluster) +  
  geom_density2d() +  
  theme_bw() +  
  labs(title = "Время ожидания между извержениями и продолжитель-  
ность извержения",  
        subtitle = "Старый Верный гейзер, штат Вайоминг, США.",  
        x = "Продолжительность извержения, минуты",  
        y = "Ожидание до следующего извержения, минуты") +  
  scale_x_continuous(breaks = seq(min(df$eruptions), max(df$eruptions),  
0.25)) +  
  scale_y_continuous(breaks = seq(min(df$waiting), max(df$waiting), 5))
```

На рисунке 63 представлен пример графика.

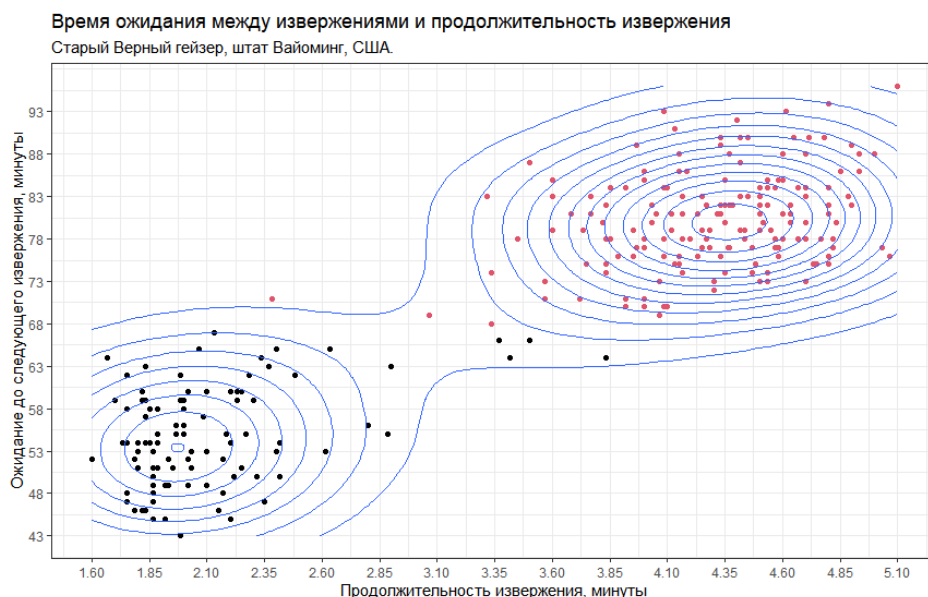


Рисунок 63 Красивая графика это просто

Отрисовка временного ряда

Чтобы научиться настраивать графику с помощью библиотеки `ggplot2` необходимо несколько раз проделать некоторые базовые вещи чтобы понять суть и концепции данного инструмента. После нескольких повторений данная процедура будет занимать минимум времени и будет приносить максимум пользы.

Для того чтобы полностью погрузиться в работу над графикой, нам необходимо проделать весь процесс в рамках поставленных условий, это поможет нам не сбиться с маршрута и не наплодить лишних графических деталей.

К графикам временного ряда в анализе предъявляются некоторые требования, которые помогают процессу анализа:

- график должен быть квадратным;
- график временного ряда должен занимать всю полезную площадь поверхности рисования;
- график временного ряда должен быть отражён в виде линий с маркерами, где каждая точка будет различимой на фоне линий;
- подписи к графику должны быть понятными, единицы измерения величин единообразно определяемыми.

После того, как мы определились с ограничениями к графику временного ряда, необходимо определиться с рядом данных. Воспользуемся скриптом загрузки финансовых данных из самостоятельной работы №6. Будем изучать графику временных рядов вместе с фреймом данных `df`.

Квадратный график. Сохранение рисунков определённого размера в

R производится путём сохранения содержимого графического устройства сессии в виде изображения одного из популярных форматов. Одним из самых популярных форматов является формат .png. Покажем, каким образом можно в R сохранять отдельные файлы изображений в формате png:

```
# Открытие png устройства
png(filename = "timeseries.png", width = 800, height = 800, units = "px")
# Функции графика
dev.off() # Сохранить в папке
```

Функция png() и dev.off() окаймляют отрисовку графика. Аргументы функции png() в комментариях не нуждаются.

Полезная площадь рисования. Для разбирательств с полезной площадью графика давайте попробуем хотя бы построить обычный график ряда. В функции ggplot() необходимо указать источник данных и оси координат, чтобы определить холст, также чтобы определить линии и точки необходимо добавить также некоторые слои через “+”:

```
# Открытие png устройства
png(filename = "timeseries.png", width = 800, height = 800, units = "px")
ggplot(data = df, mapping = aes(x = 1:nrow(df), y = df[[4]])) +
  geom_line() +
  geom_point()
dev.off() # Сохранить в папке
```

В качестве абсцисс мы указали перечисления торговых дней с момента начала торгов по акциям, до собственно конца строк в таблиц. В качестве ординат была взята четвёртая колонка фрейма данных. Функции geom_line() и geom_point() позволили отобразить точки на пересечении данных абсцисс-ординат, а также протянуть между точками прямые линии.

В результате выполнения, у нас возник в папке проекта файл “timeseries.png”, который выглядит следующим образом:

На рисунке 64 показан обычный график временного ряда.

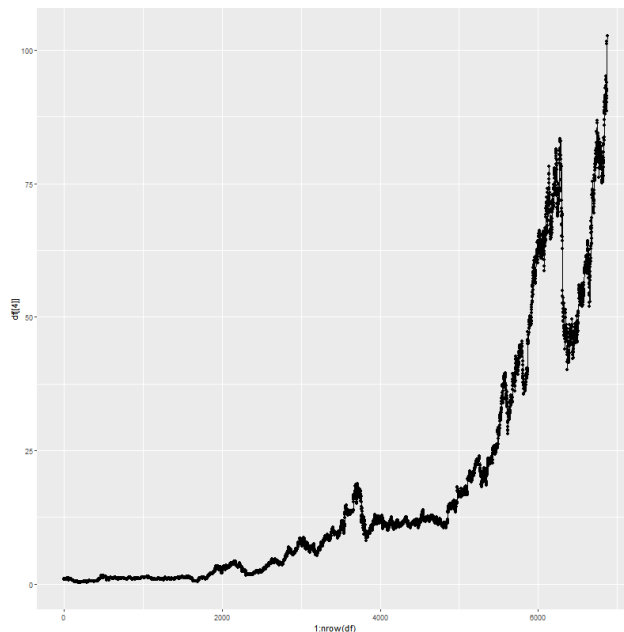


Рисунок 64 Обычный график временного ряда

Полезная площадь рисования графика определяется сеткой на которой график рисуется. Подрезать границы графика в случае образования пустого места можно с помощью `coord_cartesian()`, функция принимает границы рисования по `x` и `y` в аргументы `xlim` и `ylim` соответственно:

```
# Открытие png устройства
png(filename = "timeseries.png", width = 800, height = 800, units = "px")

ggplot(data = df, mapping = aes(x = 1:nrow(df), y = df[[4]])) +
  geom_line() +
  geom_point() +
  coord_cartesian(xlim = c(1, nrow(df)), ylim = range(df[[4]]))

dev.off() # Сохранить в папке
```

Различные точки. В соответствующих слоях для отображения геометрии рисунка можно настроить некоторые особенности этих слоёв: размер, цвет, толщина, штрих-пунктир, прозрачность и т.д.

Изменим на нашем графике толщину линий и маркеров для соблюдения третьего требования:

```
ggplot(data = df, mapping = aes(x = 1:nrow(df), y = df[[4]])) +
  geom_line(lwd = I(0.5), lty = 1) +
  geom_point(cex = I(0.5)) +
  coord_cartesian(xlim = c(1, nrow(df)), ylim = range(df[[4]]))
```

В аргументах `geom_` мы использовали аргументы `lwd`, `lty` и `cex`. Это

стандартные для линий и точек аргументы графики. Аргумент `lwd` отвечает за толщину линии, значение `I(0.5)` обозначает толщину вдвое меньшую стандартной. Аргумент `lty` – тип штрих-пунктира линии, определён целыми числами от 1 до 5. Аргумент `cex` – стандартный параметр точек, обозначает величину точек.

Подписи к графику. В библиотеке `ggplot2` подписи к графикам ставятся через функцию `labs()`:

```
ggplot(data = df, mapping = aes(x = 1:nrow(df), y = df[[4]])) +  
  geom_line(lwd = I(0.5), lty = 1) +  
  geom_point(cex = I(0.5)) +  
  coord_cartesian(xlim = c(1, nrow(df)), ylim = range(df[[4]])) +  
  labs(title = "График цен акций компании Activision-Blizzard",  
        subtitle = paste("Данные от", min(rownames(df))),  
        x = "Торговые дни от начала торгов, отсчитанные с 1",  
        y = "Цены акций в долл.США")
```

Добавляем деталей. Цвет холста – дело вкуса. Поменять цвет холста в `ggplot2` на белый можно с помощью `theme_bw()`.

Изменить дискретизация подписей координатной сетки можно с помощью `scale_x_continuous()`

Открытие png устройства

```
png(filename = "timeseries.png", width = 800, height = 800, units = "px")  
  
ggplot(data = df, mapping = aes(x = 1:nrow(df), y = df[[4]])) +  
  geom_line(lwd = I(0.5), lty = 1) +  
  geom_point(cex = I(0.5)) +  
  coord_cartesian(xlim = c(1, nrow(df)), ylim = range(df[[4]])) +  
  labs(title = "График цен акций компании Activision-Blizzard",  
        subtitle = paste("Данные от", min(rownames(df))),  
        x = "Торговые дни от начала торгов, отсчитанные с 1",  
        y = "Цены акций в долл.США") +  
  scale_x_continuous(breaks = seq(1, nrow(df), 365)) +  
  scale_y_continuous(breaks = seq(0, 100, 10)) +  
  theme_bw()
```

`dev.off()` # Сохранить в папке

На рисунке 65 показан график временного ряда по требованиям.

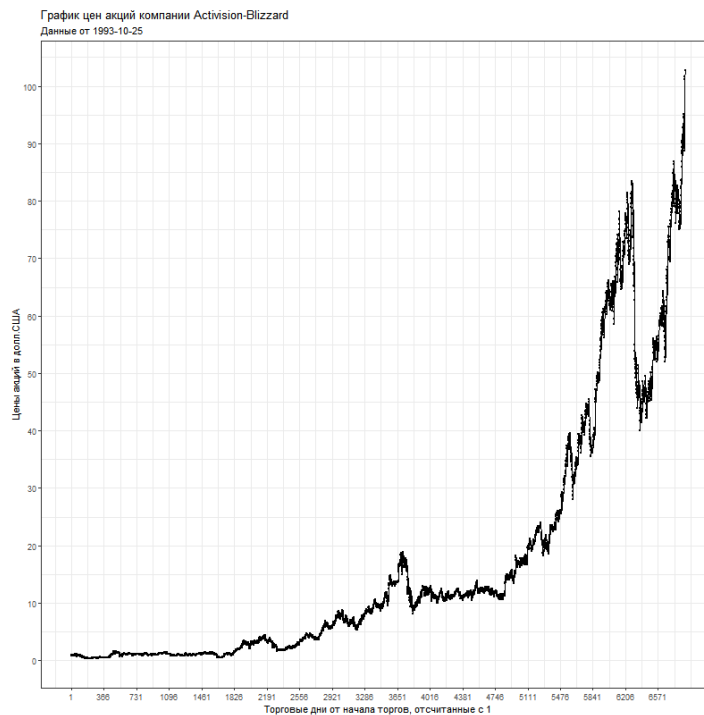


Рисунок 65 График временного ряда по требованиям

Для отображения гистограммы ряда необходимо произвести некоторые изменения в коде, поменять слои и варианты отображения:

Открытие png устройства

```
png(filename = "histogram.png", width = 600, height = 600, units = "px")
```

```
ggplot(data = df, mapping = aes(df[[4]])) +
```

```
  geom_histogram(bins = 40, color = "black", fill = "grey") +
```

```
  geom_freqpoly(bins = 40, color = "red4", lwd = I(1.1)) +
```

```
  labs(title = "Гистограмма распределения данных статического разреза",
```

```
        subtitle = "Временной ряд цен акций Activision Blizzard",
```

```
        x = "Данные цен акций в долл.США",
```

```
        y = "Количество измерений в промежутке") +
```

```
  scale_x_continuous(n.breaks = 15) +
```

```
  scale_y_continuous(n.breaks = 15) +
```

```
  theme_bw()
```

```
dev.off() # Сохранить в папке
```

В данном скрипте выбраны следующие слои для отображения гистограммы. Для построения столбцов был использован слой `geom_histogram()`. Для построения линии, соединяющей середины отрезка `geom_freqpoly()`. Остальные параметры были использованы ранее.

Для определения данных в `aes()` для гистограммы необходимы только сами данные без оси абсцисс.

На рисунке 66 показана гистограмма распределения измерений по естественным границам значений ряда.

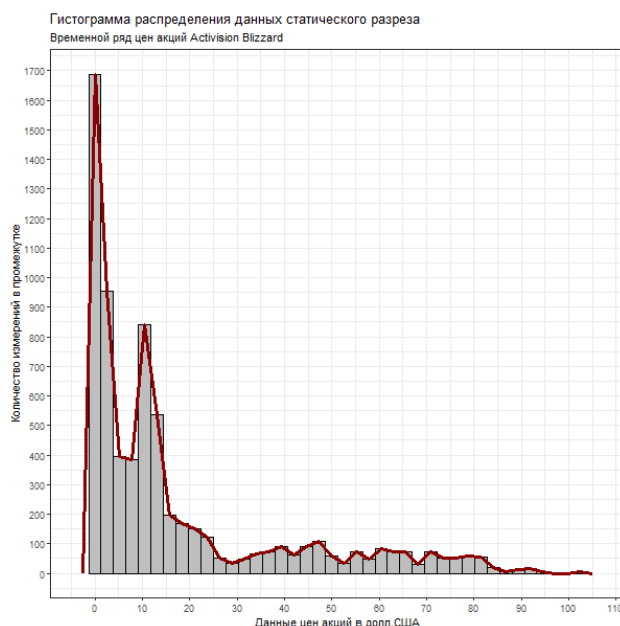


Рисунок 66 Гистограмма распределения измерений по естественным границам значений ряда

Для отображения вероятностей и ядровой функции распределения используются хитрые приёмы, которые просто лучше запомнить:

```
# Открытие png устройства
```

```
png(filename = "density_histogram.png", width = 600, height = 600, units = "px")
```

```
ggplot(data = df, mapping = aes(df[[4]])) +  
  geom_histogram(aes(y = ..density..),
```

```
    binwidth=density(df[[4]])$bw,
```

```
    color = "black",
```

```
    fill = "grey") +
```

```
  geom_density(fill="red", alpha = 0.2)+
```

```
  labs(title = "Гистограмма и ядровая функция распределения данных",
```

```
        subtitle = "Временной ряд цен акций Activision Blizzard",
```

```
        x = "Данные цен акций в долл.США",
```

```
        y = "Вероятность попадания значения в промежуток") +
```

```
  scale_x_continuous(n.breaks = 15) +
```

```
  scale_y_continuous(n.breaks = 15) +
```

```
  theme_bw()
```

```
dev.off() # Сохранить в папке
```

На рисунке 67 показан результат выполнения данных команд – файл с визуализацией:

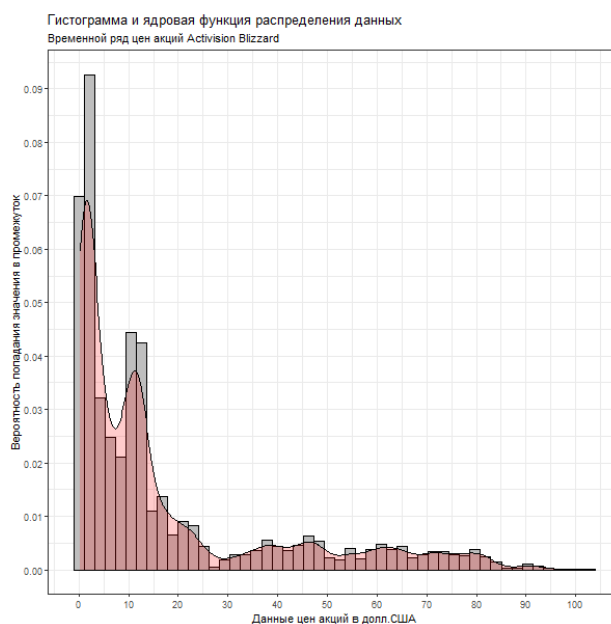


Рисунок 67 Гистограмма вероятностей и ядерная функция распределения для данных статического разреза

Самостоятельная работа №10

Часть 1

Задание 1.

Все графики строятся с помощью библиотеки ggplot2.

1. Загрузите файл `demography.csv`. В нём содержатся данные по населению Белгородской и Калужской областей за 2016 год (источник – Росстат).

Переменные:

- `region`: название региона;
- `district`: название района;
- `empl_total`: численность занятого населения;
- `A-O`: занятость по отраслям (как на сайте Росстата: сельское хозяйство,);
- `popul_total`: численность населения;
- `urban_total`: численность городского населения;
- `rural_total`: численность сельского населения;
- `wa_total`: численность трудоспособного населения;
- `wa_female`: численность трудоспособного населения (женский пол);

- `wa_male`: численность трудоспособного населения (мужской пол);
 - `ret_total`: численность пенсионеров;
 - `ret_female`: численность пенсионеров (женский пол);
 - `ret_male`: численность пенсионеров (мужской пол);
 - `young_total`: численность населения, моложе трудоспособного возраста;
 - `young_female`: численность населения, моложе трудоспособного возраста (женский пол);
 - `young_male`: численность населения, моложе трудоспособного возраста (мужской пол);
 - `X18_19 – X70_plus`: численность населения по возрастным группам.
2. Создайте переменную `young_share` – процент населения возраста, моложе трудоспособного. Создайте переменную `trud_share` – процент населения трудоспособного возраста и `old_share` – процент населения возраста, старше трудоспособного.
 3. Постройте гистограмму для доли трудоспособного населения в процентах. Измените цвет гистограммы, добавьте *rugs*. Добавьте вертикальную линию, которая очерчивает медианное значение доли трудоспособного населения в процентах.
 4. Постройте сглаженные графики плотности распределения для доли трудоспособного населения в процентах по регионам (два графика в одной плоскости). Настройте цвета и прозрачность заливки. По графикам плотности определите, имеет ли смысл для визуализации распределения доли трудоспособного населения строить скрипичные диаграммы (*violin plot*). Если да, постройте их (так же по группам). Если нет, постройте ящики с усами.
 5. Постройте диаграмму рассеяния для переменных `young_share` и `old_share`. Можно ли сказать, что чем больше процент молодого населения (моложе трудоспособного населения), тем меньше процент пожилых людей (старше трудоспособного возраста)? Поменяйте цвет и тип маркера для точек.
 6. Создайте переменную `male_share` – доля мужского населения в районе/городе (в процентах). Создайте переменную `male`, которая принимает значение 1, если доля мужчин в муниципальном районе/городе больше доли женщин, и значение 0 – во всех остальных случаях.

7. Постройте пузырьковую диаграмму (*bubble plot*) для переменных *young_share* и *old_share*, учитывая информацию о доле мужчин в районе и о том, преобладают ли мужчины в районе или нет.

Постройте столбиковую диаграмму (*bar plot*), которая показывала бы, сколько в базе данных районов Белгородской области, а сколько – Калужской.

Часть 2

Задание 1

В данном задании нужно работать со встроенной в R базой данных по автомобилям *mtcars*. Загружать ее по ссылке не нужно, достаточно набрать ее название (`data = mtcars`). Например, чтобы посмотреть на базу, можно просто воспользоваться `View(mtcars)`.

Постройте с помощью библиотеки *ggplot2* пузырьковую диаграмму (*bubble plot*), которая

- показывала бы связь между показателями *Gross horsepower* (*hp*) и *Weight* (*wt*);
- учитывала бы информацию о числе цилиндров у автомобиля (*cyl*);
- учитывала бы информацию о типе коробки передач – автоматическая или нет (*am*); сделайте так, чтобы легенда графика была корректной и информативной + пусть точки, соответствующие автомобилям с автоматической коробкой передач, будут зеленого цвета ("green"), а с ручной – красного ("red").

Подпишите оси (дайте им более вразумительные названия). Добавьте название (заголовок) графика.

Задание 2

Работая с той же базой *mtcars*, воспроизведите следующий график:

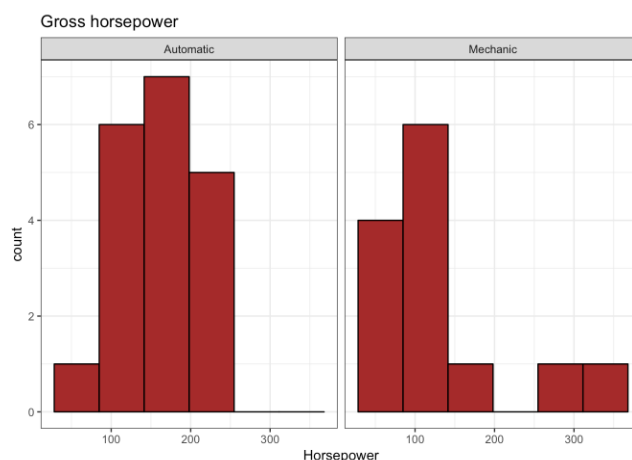


Рисунок 68 Пример графика

Подсказка: цвет – “brown”, 0 – автоматическая коробка передач, 1 – ручная (am) число столбцов (bins) можно определить по графику.

Задание 3

В данном задании нужно работать с базой sleep, встроенной в R.

Постройте «ящики с усами» в пределах одной области для графика, которые иллюстрировали бы распределение переменной extra по группам испытуемых. Поменяйте базовые цвета заливки графиков, добавьте подписи к осям и заголовок графика.

Часть 3

Визуализация данных временного ряда COVID19

В данной работе для проработки изложенного материала предлагается визуализировать данные временного ряда, полученные ранее в ходе выполнения самостоятельной работы №3 (предобработка данных Covid19).

Задание

Прodelать все шаги визуализации данных временного ряда, рассмотренные в практической работе №10 применительно к 3 временным рядам различных стран (на выбор) из фрейма данных полученного в самостоятельной работе №3.

Практическая работа №11

ООП в R и S3 классы

Свойства классов

Каждый объект в языке R является классом. Каждый класс, в зависимости от реализации, имеет свой набор полей и методов, с которыми он функционирует. В терминах объектно-ориентированного программирования (ООП) возможность объединения схожих по набору свойств и методов объектов в группы (классы) называется инкапсуляция.

Вектор – элементарная структура данных в R и имеет минимальное число полей. Вектор он обладает таким свойством как длина (length). Для примера мы возьмём встроенный вектор letters.

```
> length(letters)
[1] 26
```

С помощью функции length мы получили длину вектора letters. Теперь попробуем применить эту же функцию к встроенному дата фрейму iris.

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

Функция length, применимая к таблицам, возвращает количество столбцов.

У таблиц есть и другое свойство, размерность.

```
> dim(iris)
[1] 150 5
```

Функция dim в примере выше выводит информацию о том, что в таблице iris 150 строк и 5 столбцов.

В свою очередь, у вектора нет размерности.

```
> dim(letters)
NULL
```

Таким образом, мы убедились, что у объектов разного класса имеется разный набор свойств.

Обобщённые функции

В R множество обобщённых функций: print, plot, summary и т.д. Эти функции по-разному работают с объектами разных классов.

Возьмём, например, функцию plot и запустим её, передав в качестве её главного аргумента таблицу iris.

```
> plot(iris)
```

На рисунке 69 показан стандартный вызов функции `plot()` для встроенной таблицы данных `iris`.

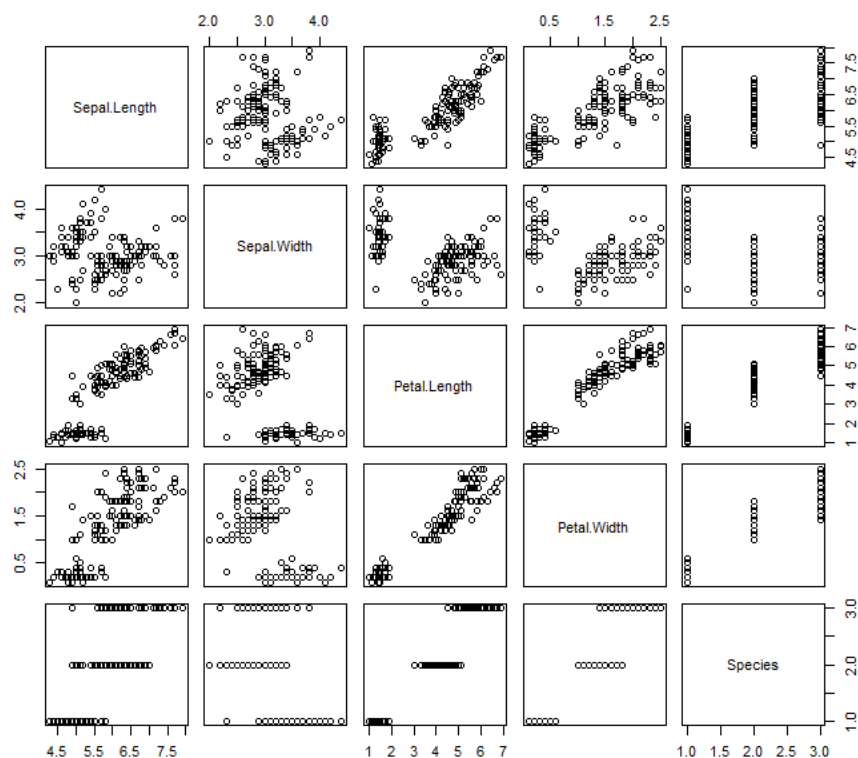


Рисунок 69 Стандартный вызов функции `plot()` для встроенной таблицы данных `iris`

Попробуем отобразить теперь при помощи функции `plot()` вектор случайных значений из равномерного распределения:

```
> plot(runif(100, min = -3, max = 3))
```

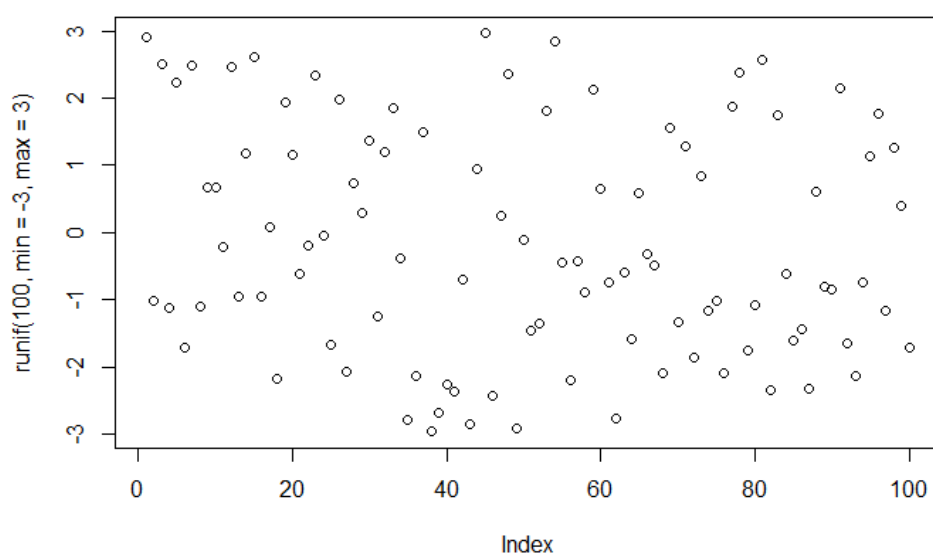


Рисунок 70. Стандартное отображение вектора при использовании функции `plot()`

Мы получили разные графики, в первом случае корреляционную

матрицу, во втором – график рассеивания, на котором по оси x отображается индекс наблюдения, а по оси y его значение.

Таким образом, функция `plot` умеет подстраиваться под работу с разными классами. Если вернуться к терминологии ООП, то возможность определить класс входящего объекта и выполнять различные действия с объектами разных классов называется полиморфизм.

Получается это за счёт того, что данная функция всего лишь является оболочкой к множеству методов, написанных под работу с разными классами. Убедиться в этом можно с помощью следующей команды:

```
> body("plot")
UseMethod("plot")
```

Команда `body` выводит в консоль R тело функции. Как видите тело функции `body` состоит всего из одной команды `UseMethod("plot")`.

Таким образом, функция `plot`, лишь запускает один из множества написанных к ней методов в зависимости от класса передаваемого ей объекта. Посмотреть список всех её методов можно следующим образом.

```
> methods("plot")
[1] plot.acf*      plot.data.frame*  plot.decomposed.ts*
[4] plot.default   plot.dendrogram*  plot.density*
[7] plot.ecdf      plot.factor*      plot.formula*
[10] plot.function  plot.hclust*      plot.histogram*
[13] plot.HoltWinters* plot.isoreg*      plot.lm*
[16] plot.medpolish* plot.mlm*         plot.ppr*
[19] plot.prcomp*   plot.princomp*    plot.profile.nls*
[22] plot.raster*   plot.spec*        plot.stepfun
[25] plot.stl*      plot.table*       plot.ts
[28] plot.tskernel* plot.TukeyHSD*
see '?methods' for accessing help and source code
```

Полученный результат говорит о том, что функция `plot` имеет 29 методов, среди которых есть `plot.default`, который срабатывает по умолчанию, если функция получает на вход объект неизвестного ей класса.

С помощью функции `methods` также можно получить и набор всех обобщённых функций, у которых есть метод, написанный под какой-либо класс.

```
> methods(, "matrix")
[1] anyDuplicated as.data.frame as.raster boxplot
[5] coerce determinant duplicated edit
```

```
[9] head      initialize  isSymmetric  Math
[13] Math2      Ops        relist      subset
[17] summary    tail        unique
see '?methods' for accessing help and source code
```

Что такое S3 класс и как создать собственный класс

В R есть ряд классов, которые вы можете создавать самостоятельно. Один из наиболее популярных – S3.

Данный класс представляет собой список, в котором хранятся различные свойства созданного вами класса. Для создания собственного класса достаточно создать list и присвоить ему название класса.

```
# создаём структуру класса
employee1 <- list(name      = "Oleg",
                  surname    = "Petrov",
                  salary      = 1500,
                  salary_datetime = Sys.Date(),
                  previous_salary = NULL,
                  update      = Sys.time())
```

```
# присваиваем объекту класс
class(employee1) <- "emp"
```

Таким образом, мы создали свой собственный класс, который в своей структуре хранит следующие данные:

- Имя сотрудника.
- Фамилия сотрудника.
- Зарплата.
- Время, когда была установлена зарплата.
- Предыдущая зарплата.
- Дата и время последнего обновления информации.

После чего командой `class(employee1) <- "emp"` мы присваиваем объекту класс `emp`. Для удобства создания объектов класса `emp` можно написать функцию.

```
# функция для создания объекта
create_employee <- function(name,
                             surname,
                             salary,
                             salary_datetime = Sys.Date(),
```

```

        update      = Sys.time()) {
out <- list(name      = name,
            surname   = surname,
            salary     = salary,
            salary_datetime = salary_datetime,
            previous_salary = NULL,
            update     = update)

class(out) <- "emp"

return(out)
}

# создаём объект класса emp с помощью функции create_employee
employee1 <- create_employee("Oleg", "Petrov", 1500)

# проверяем класс созданного объекта
class(employee1)

```

Функции присваивания значений пользовательским S3 классам

Итак, мы создали собственный класс emp, но пока это нам ничего не дало. Покажем, зачем мы создали свой класс и что с ним можно делать.

В первую очередь вы можете написать функции присваивания для созданного класса.

Функция присваивания для [.

```

"[<-.emp" <- function(x, i, value) {
  if ( i == "salary" || i == 3 ) {
    cat(x$name, x$surname, "has changed salary from", x$salary, "to", value)
    x$previous_salary <- x$salary
    x$salary          <- value
    x$salary_datetime <- Sys.Date()
    x$update          <- Sys.time()
  } else {
    cat( "You can't change anything except salary" )
  }
  return(x)
}

```

Функция присваивания для [].

```
"[<-.emp" <- function(x, i, value) {  
  
  if ( i == "salary" || i == 3 ) {  
    cat(x$name, x$surname, "has changed salary from", x$salary, "to", value)  
    x$previous_salary <- x$salary  
    x$salary          <- value  
    x$salary_datetime <- Sys.Date()  
    x$update          <- Sys.time()  
  } else {  
    cat( "You can`t change anything except salary" )  
  }  
  return(x)  
}
```

При создании функции присваивания всегда указываются в кавычках, и имею вид:

```
"[<-.имя класса" / "[<-.имя класса"
```

А также имеют три обязательных аргумента.

- `x` – объект, которому будет присваиваться значение;
- `i` – имя / индекс элемента объекта (`name`, `surname`, `salary`, `salary_datetime`, `previous_salary`, `update`);
- `value` – присваиваемое значение.

Далее в теле функции вы пишете, как должны измениться элементы вашего класса. В нашем случае мы хотим, чтобы у пользователя была возможность менять только зарплату (элемент `salary`, индекс которого равен 3). Поэтому внутри функции мы пишем проверку `if (i == "salary" || i == 3)`. В случае если пользователь пытается редактировать другие свойства, он получает сообщение "You can't change anything except salary".

При изменении элемента `salary` выводится сообщение, содержащее имя и фамилию сотрудника, его текущий и новый уровень зарплаты. Текущая зарплата передаётся в свойство `previous_salary`, а `salary` присваивается новое значение. Также обновляются значения свойств `salary_datetime` и `update`.

Теперь можно попытаемся изменить зарплату.

```
> employee1["salary"] <- 1750
```


Oleg Petrov has changed salary from 1500 to 1750

Разработка собственных методов для обобщённых функций

Ранее вы уже узнали, что в R существуют обобщённые функции, которые меняют своё поведение в зависимости от класса, получаемого на вход объекта. Вы можете дописывать свои методы существующим обобщённым функциям и даже создавать свои обобщённые функции. Одной из наиболее часто используемых обобщённых функций является `print`. Данная функция срабатывает каждый раз, когда вы вызываете объект по его названию. Сейчас вывод на печать созданного нами объекта класса `emp` имеет вид:

```
$name
[1] "Oleg"
$surname
[1] "Petrov"
$salary
[1] 1750
$salary_datetime
[1] "2021-02-09"
$previous_salary
[1] 1500
$update
[1] "2021-02-09 22:13:43 MSK"
attr("class")
[1] "emp"
```

Давайте напишем свой метод для функции `print`.

```
print.emp <- function(x) {
  cat("Name:", x$name, x$surname, "\n",
      "Current salary:", x$salary, "\n",
      "Days from last update:", Sys.Date() - x$salary_datetime, "\n",
      "Previous salary:", x$previous_salary)
}
```

Теперь функция `print` умеет выводить на печать объекты нашего самодписного класса `emp`. Достаточно просто ввести в консоль имя объекта и получим следующий вывод.

```
>print(employee1)
Name: Oleg Petrov
Current salary: 1750
```

Days from last update: 0

Previous salary: 1500

Создание обобщённой функции и методов

Большинство обобщённых функций внутри выглядят однотипно и просто используют функцию UseMethod.

```
# обобщённая функция
get_salary <- function(x, ...) {
  UseMethod("get_salary")
}
```

Теперь напомним для неё два метода, один для работы с объектами класса emp, второй метод будет запускаться по умолчанию для объектов всех других классов, под работу с которыми у нашей обобщённой функции нет отдельно написанного метода.

```
# метод для обработки объектов класса emp
get_salary.emp <- function(x) x$salary
# метод который срабатывает по умолчанию
get_salary.default <- function(x) cat("Work only with emp class objects")
```

Название метода состоит из имени функции и класса объектов, которые данный метод будет обрабатывать. Метод default будет запускаться каждый раз, если вы передаёте в функцию объект класса, под который не написан свой метод.

```
> get_salary(employee1)
[1] 1750
> get_salary(iris)
Work only with emp class objects
```

Наследование

В нашем примере мы можем выделить в отдельный подкласс remote_emp сотрудников, работающих удалённо. Такие сотрудники будут иметь дополнительное свойство: город проживания.

```
# создаём структуру подкласса
employee2 <- list(name      = "Ivan",
                  surname   = "Ivanov",
                  salary     = 500,
                  salary_datetime = Sys.Date(),
```

```

previous_salary = NULL,
update          = Sys.time(),
city            = "Moscow")

# присваиваем объекту подкласс remote_emp
class(employee2) <- c("remote_emp", "emp")

# проверяем класс объекта
class(employee2)
[1] "remote_emp" "emp"

```

При операции присваивании класса, создавая подкласс, мы используем вектор, в котором первым элементом идёт имя подкласса, далее – имя родительского класса.

В случае наследования все обобщённые функции и методы, написанные для работы с родительским классом, будут корректно работать и с его подклассами.

```

> print(employee2)
Name: Ivan Ivanov
Current salary: 500
Days from last update: 0
Previous salary:
> get_salary(employee2)
[1] 500

```

Но вы можете разрабатывать методы отдельно для каждого подкласса.

```

# метод для получения свойства salary объектов подкласса remote_emp
get_salary.remote_emp <- function(x) {
  cat(x$surname, "remote from", x$city, "\n")
  return(x$salary)
}

# запрашиваем свойство salary объекта подкласса remote_emp
get_salary(employee2)
Ivanov remote from Moscow
[1] 500

```

Работает это следующим образом. Сначала обобщённая функция ищет метод написанный для подкласса `remote_emp`, если не находит то идёт дальше и ищет метод написанный для родительского класса `emp`.

Список использованной литературы

1. Венэблз У.Н., Смит Д. М. и Рабочая группа разработки R. Введение в R. Заметки по R: среда программирования для анализа данных и графики. – М.: 2012. – 109 с.
2. Золотарюк А.В. Язык и среда программирования R. Учебное пособие. – ИНФРА-М, 2019. – 162 с.
3. Лонг Дж., Титор П. Р. Книга рецептов. Изд. ДМК, Москва, 2020. – 511 с.
4. Мастицкий С.Э., Шитиков В.К. Статистический анализ и визуализация данных с помощью R. Хайдельберг – Лондон – Тольятти, 2014. – 401 с.
5. Мастицкий С. Э. Работа с RStudio. Перевод официального руководства пользователя [Электронный ресурс]. – URL: <https://docviewer.yandex.ru/view/3158527> (Дата обращения 23.03.2020)
6. Шипунов А.Б., Е.М. Балдин, П.А. Волкова и др.: Наглядная статистика. Используем R! ДМК Пресс, 2014. – 298 с.
7. Шитиков В. К., Мастицкий С. Э. (2017) Классификация, регрессия и другие алгоритмы Data Mining с использованием R [Электронный ресурс]. – URL: <https://github.com/ranalytics/data-mining> (Дата обращения 23.03.2020)

Обзор ресурсов данных для анализа

1. Обзор разных баз:
<https://www.dataquest.io/blog/free-datasets-for-projects/>
2. Открытые данные по России: <http://data.gov.ru/>
3. HubOfData: <https://hubofdata.ru/dataset>
4. Github с открытыми базами по разным темам:
<https://github.com/awesomedata/awesome-public-datasets>

Учебные базы, встроенные в разные библиотеки R, но доступные вне его:
<https://vincentarelbundock.github.io/Rdatasets/>