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 доступных;

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, перечисленныхв табл. 1.

Таблица 1

Функции вызова справки в R

Функция	Действие
help.start()	Общая справка
help (–*предмет*‖) или ? *предмет*	Справка по функции *предмет* (кавычки необязательны)
help.search (–*предмет*‖) или ??*предмет*	Поиск в справке записей, содержащих *предмет*
example(–*предмет* ‖)	Примеры использования функции *предмет* (кавычки необязательны)
RSiteSearch(–*предмет* ‖)	Поиск записей, содержащих *предмет* в онлайн-руководствах и заархивированных рассылках
apropos(–*предмет* ‖, mode=‖function‖)	Список всех доступных функций, в названии которых есть *предмет*
data()	Список всех демонстрационных данных, содержащихся в загруженных пакетах
vignette()	Список всех доступных руководств по загруженным пакетам
vignette(–*предмет* ‖)	Список руководств по теме *предмет*
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,"clipb oard",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.package(). 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
```

e 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 присутствует 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 тесктовых 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 переменой.

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:

```
 Черный 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] " Черный 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 – пригласительную 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 результата
```

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.

```
# пример 1
a<-3
b<-6
vect1<-a:b # пример 1
vect1
# пример 2
m<--3
n<-6
vect2<-m:n # пример 1
vect2
# пример 3
vect3<-1:100
vect3
# пример 4
vect4<-c(-8,5,0,45,11)
vect4
# пример 5
vect5<-c(-5:5)
vect5
```

Рисунок 2 Варианты создания векторов

Векторы с текстовыми данными могут быть созданы несколькими способами:

```
# пример 1
vect1<-c("Москва","Екатеринбуог","Сочи","Ростов")
vect1
# пример 2
vect2<-c(-1:5,10.5,"next")
vect2
vect2[6]
```

Рисунок 3 Создание векторов с текстовыми данными

Имеются возможностей для создания векторов из вещественных чисел:

```
# пример 1
vect1<-2.5:9.6
vect1
# пример 2
vect2<-seq(5,10,by=0.4)
vect2
vect2[3]# вывод 3-его элемента вектора
f<-1.55
vect3<-seq(-2.13,12.5,by=f)
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<-comp1ex(5) s[1]<-4.123456769+21 s[l] [1] 4.123457+21
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.

```
# пример 1 Сумма векторов
vect1<-c(2,3,4,5)
vect1
vest2<-seq(-3,3)
vect2
sum<-vect1+vect2
sum
#пример 2 Разность векторов
vect1<-c(2,3,4,5)
vect1
vest2<-seq(-3,3)
vect2
razn<-vect1-vect2
razn
#пример 3 Произведение векторов
vect1<-c(2,3,4,5)
vect1
vest2<-seq(-3,3)
vect2
pr<-vect1*vect2
pr
```

```
#пример 4 Деление векторов
vect1<-c(2,3,4,5)
vect1
vest2<-seq(-3,3)
vect2
del<-vect1/vect2
del
#пример 5 Возведение в степень
vect1<-2
vect1
vect2<-1:10
vect2
voz<-vect2^vect1
voz
#пример 6 Изменение знака вектора
vect2<-1:10
vect1<--vect2
vect1
```

Приведем примеры сравнения векторов.

```
# Сравнение векторов на операцию не больше
vect1<-c(-8,0,5,11)
vect1
vect2<-c(-3,-2,-1,0,1,2,3,4)
vect2
sr<-vect2<=vect1
sr
# Сравнение векторов на операцию не равно
vect1<-c(-8,0,5,11)
vect1
vect2<-c(-3,-2,-1,0,1,2,3,4)
vect2
sr<-vect2!=vect1
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:

```
# Сортировка вектора по возрастанию
x <- c(174, 162, 188, 192, 165, 168, 172)
sort_1<-sort(x,decreasing = FALSE)
sort_1
# Сортировка вектора по убыванию
x <- c(174, 162, 188, 192, 165, 168, 172)
sort_1<-sort(x,decreasing = TRUE)
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.

```
vect1<-c("Антонов А.А.","Борисов Б.Б.","Воронин В.В.","Гусев Г.Г.",
         "Дмитриев Д.Д.","Ерохин Е.Е.","Жук С.А.")
vect2<-c("Жук С.А.","Петров П.П.","Антонов А.А.","Романов Р.Р.")
vect1
vect2
# Оценка принадлежности элементов более короткого вектора более длинному
rez<-vect2 %in% vect1
rez
# Оценка принадлежности элементов более длинного вектора более короткому
rez<-vect1 %in% vect2
rez
# Сравнение векторов на неравенство
rez2<-vect1!=vect2
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().

```
vect1<-c("Понедельник","Вторник","Среда","Четверг","Пятница")
vect1
vect2<-c("Анализ данных","Информатика","математика","Стптистика",
         "Теория вероятности")
vect2
int5<-c(2,4,4,2)# количество часв занятий
t<-"час."
# 1 вариант Вывод единой строки
rez<-paste(vect1,vect2,int5,t,sep=" - ",collapse=";")
rez
rez[2]
#  вариант Поэлементный вывод результата
rez<-paste(vect1,vect2,int5,t,sep=" - ")
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 заменям, 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 начений 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 сработает, 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 <= 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