Лекція 12.
Ймовірнісне програмування
Мультипарадигмена мова програмування R

Что такое R?

- Программное средство для
  - Чтения и манипулирования данными
  - Вычислений
  - Проведения статистического анализа
  - Отображения результатов
- Реализация языка S – языка для манипулирования объектами
- Среда программирования

  R – это среда программирования для анализа данных и графики. Язык изначально был создан Ross Ihaka и Robert Gentleman на кафедре статистики в университете Окленда. Сейчас множество людей развивают этот язык.
- Платформа для разработки и внедрения новых алгоритмов.
  R предоставляет платформу для разработки новых алгоритмов и передачи методов. Это может быть достигнуто тремя путями:
  - Функции, которые используют существующие в R алгоритмы
  - Функции, которые вызывают процедуры, написанные на C или Fortran
  - Создание пакетов, содержащих код для обобщения и представления данных, вывода их на печать или в виде графиков

Где взять R

- **Последняя копия**

  Последняя копия R может быть скачана с вебсайта CRAN (Comprehensive R Archive Network) : *http://lib.stat.cmu.edu/R/CRAN/*.
- **R Packages**

  пакеты R также могут быть скачаны с этого сайта, или могут идти вместе с R. Список пакетов с их кратким описанием также можно найти на сайте.
- **Документация, руководство пользователя, книги**

  Тоже на этом сайте

Команды языка R

- В R все команды записываются в файл .Rhistory. Команды можно вызывать повторно, в отличие от MatLab. Чтобы убедиться в сохранении истории команд, можно явно воспользоваться функцией savedhistory(). История команд, использованных во время предыдущей сессии, может быть вызвана с помощью функции loadhistory(). Предыдущие команды вызываются клавишами ↑ и ↓.

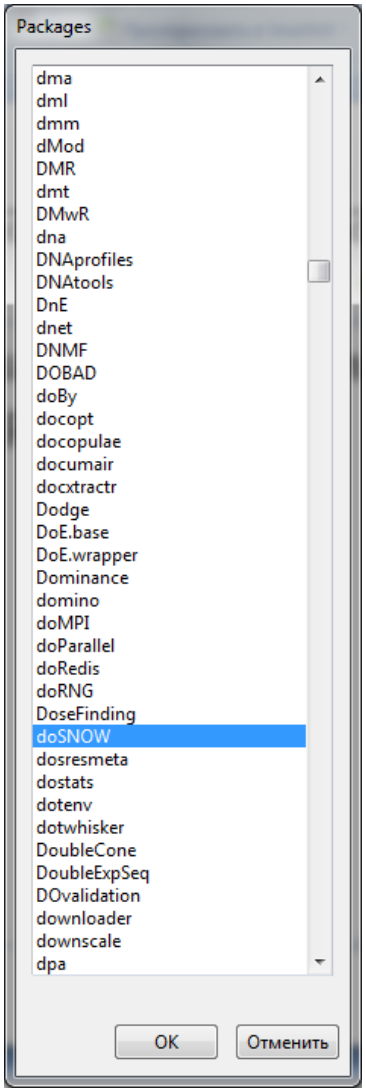
Объекты R

- По умолчанию R создает объекты в памяти и сохраняет их в единственный файл .Rdata. Объекты R автоматически сохраняются в этот файл. Пакеты загружаются в текущей сессии R.

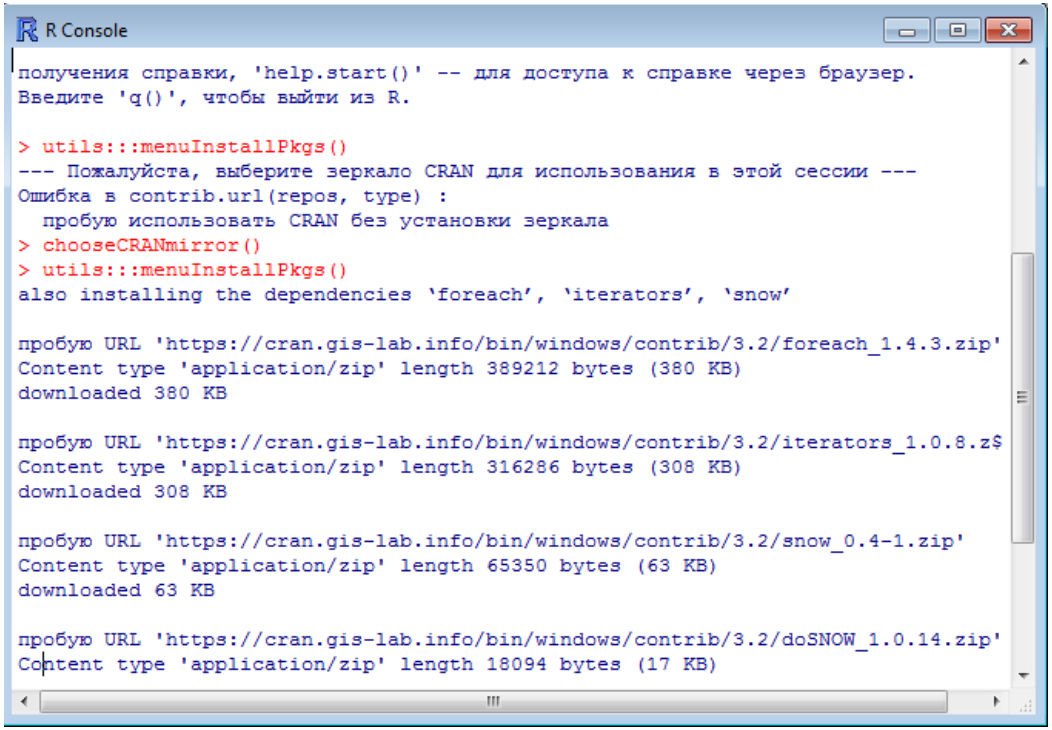
Выход из R

- Команда q()
- Или просто закрыть окно. При этом будет предложено сохранить сессию.

Инсталляция пакетов R



- Инсталлировать пакет в R можно с помощью меню Packages/Install Packages. При этом будет предложено выбрать сайт для инсталляции. После инсталляции пакеты можно загружать в R с помощью Packages/Load Package.



Язык R

Базовый синтаксис

Ввод команд в R

По умолчанию место для ввода команды в R обозначается знаком >:

```
> 5+2
[1] 7
```

Если команда синтаксически неполная, появляется знак продолжения +:

```
> 8+3*
+ 5
[1] 23
```

Оператор присваивания – левая стрелка <-:

```
> a<-4+5
> a
[1] 9
```

Ввод команд в R

Последнее выражение можно получить с помощью внутреннего объекта .Last.value:

```
> value<-.Last.value
> value
[1] 9
```

Функции rm() или remove() используются для удаления объектов из рабочей директории:

```
> rm(value)
> value
Ошибка: объект 'value' не найден
```

Имена в R

- Имена в R могут быть любыми комбинациями букв, цифр и точек, но они не могут начинаться с цифры. R чувствителен к регистру.
- Нужно избегать использования имен встроенных функций в качестве объектов. Для этого желательно проверять содержание объекта, который вы хотите использовать.

```
> value
Ошибка: объект 'value' не найден
> T
[1] TRUE
> t
function (x)
UseMethod("t")
<bytecode: 0x000000001394b0f0>
<environment: namespace:base>
```

Использование пробелов

- R игнорирует лишние пробелы между именами объектов и операторами:

```
> value <- 2 *    4
> value
[1] 8
```

- Но в операторе присваивания нельзя использовать пробел между < и -.
- Количество пробелов в выражениях, стоящих в кавычках, существенно:

```
> value<-"Hello"
> value1<-" Hello"
> value==value1
[1] FALSE
```

Справка

- Вызов справки по функции, объекту или оператору осуществляется следующими командами:

  >?function

  >help(function)

  или вызовом меню Help в R.

- Вызов справки по какой-либо теме осуществляется командой
  >help.search("topic"), например:

> help.search("linear regression")

Типы данных

- В R есть четыре атомарных типа данных
  - Numeric

    ```
    > value <- 605
    > value
    [1] 605
    ```

  - Character

    ```
    > string <- "Hello World«
    > string
    [1] "Hello World«
    ```

  - Logical

    ```
    > 2 < 4
    [1] TRUE
    ```

  - Complex number

    ```
    > cn <- 2 + 3i
    > cn
    [1] 2+3i
    ```

Атрибуты объекта

- Атрибуты важны при манипулировании объектами. У всех объектов есть два атрибута -- mode и length.

```
> mode(value)
[1] "numeric"
> length(value)
[1] 1
> mode(string)
[1] "character"
> length(string)
[1] 1
> mode(2<4)
[1] "logical"
> mode(cn)
[1] "complex"
> length(cn)
[1] 1
> mode(sin)
[1] "function"
```

- Объекты NULL – это пустые объекты без присвоенного mode. Их длина равна нулю.

```
> names(value)
[1] NULL
```

Пропущенные значения

- Во многих практических примерах некоторые элементы данных могут быть неизвестны, следовательно, им будет присвоено пропущенное значение. Код для пропущенных значений это NA. Он указывает на то, что значение элемента объекта неизвестно. Каждая операция над NA дает результат NA.
- Функция is.na() может быть использована для проверки пропущенных значений в объекте.

```
> value <- c(3,6,23,NA)
> is.na(value)
[1] FALSE FALSE FALSE TRUE
> any(is.na(value))
[1] TRUE
> na.omit(value)
[1] 3 6 23
> attr(,"na.action")
[1] 4
> attr(,"class")
[1] "omit"
```

Неопределенные и бесконечные значения

- Бесконечные и неопределенные значения (Inf, -Inf and NaN) могут быть протестированы с помощью функций is.finite, is.infinite, is.nan и is.number аналогичным образом.
- Эти значения можно получить, например, при делении на 0 или взятии логарифма от 0.

```
> value1 <- 5/0
> value2 <- log(0)
> value3 <- 0/0
> cat("value1 = ",value1," value2 = ",value2,
" value3 = ",value3,"\n")
value1 = Inf value2 = -Inf value3 = NaN
```

Арифметические операторы

оператор	описание	пример
+	Сложение	>2+5 [1] 7
-	Вычитание	>2-5 [1] -3
*	Умножение	>2*5 [1] 10
/	Деление	>2/5 [1] 0.4
^	Возведение в степень	>2^5 [1] 32
%/%	Целочисленное деление	>5%/%2 [1] 2
%%	Деление по модулю (остаток от деления)	>5%%2 [1] 1

Операторы сравнения

Оператор	описание	пример
==	равно	> value1 [1] 3 6 23 > value1==23 [1] FALSE FALSE TRUE
!=	Не равно	> value1 != 23 [1] TRUE TRUE FALSE
<	Меньше	> value1 < 6 [1] TRUE FALSE FALSE
>	Больше	> value1 > 6 [1] FALSE FALSE TRUE
<=	Меньше или равно	> value1 <= 6 [1] TRUE TRUE FALSE
>=	Больше или равно	> value1 >= 6 [1] FALSE FALSE TRUE

Логические операторы

Оператор	описание	пример
&	Поэлементное и	> value2 [1] 1 2 3 > value1==6 & value2 <= 2 [1] FALSE TRUE FALSE
|	Поэлементное или	> value1==6 | value2 <= 2 [1] TRUE TRUE FALSE
&&	Управляющее и (проверяет только первый элемент вектора)	> value1[1] <- NA > is.na(value1) && value2 == 1 [1] TRUE
||	Управляющее или (проверяет только первый элемент вектора)	> is.na(value1) || value2 == 4 [1] TRUE
xor	Поэлементное исключающее или	> xor(is.na(value1), value2 == 2) [1] TRUE TRUE FALSE
!	Логическое отрицание	> !is.na(value1) [1] FALSE TRUE TRUE

Распределения и симуляция

- В R есть множество распределений для симуляции данных, нахождения квантилей, вероятностей и функций плотности. Менее распространенные распределения находятся в специальных пакетах.
- Примеры распределений вероятности:

Функция R	Распределение	Параметры	Пакет
binom	биномиальное	size, prob	stats
exp	экспоненциальное	rate	stats
gamma	Гамма	shape, rate	stats
norm	нормальное	Mean, sd	stats

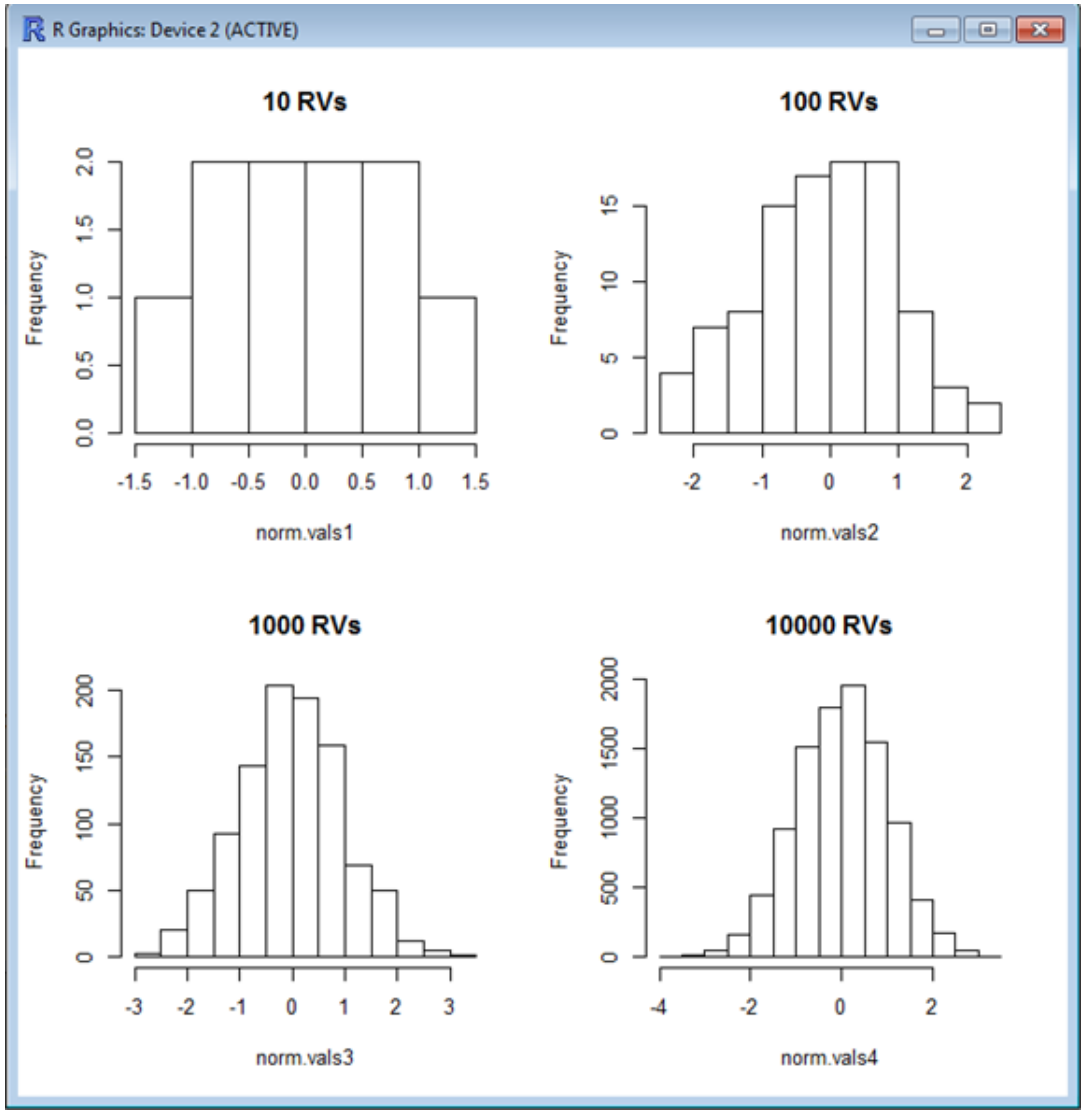
Распределения и симуляция

- В R каждое в имени каждого распределения используется префикс, обозначающий, нужно ли использовать вероятность, квантиль, функцию плотности или случайное значение. Ниже показаны все возможные префиксы:
  - p: вероятности (функции распределения)
  - q: квантили (процентные точки)
  - d: функции плотности (вероятности для дискретных случайных величин)
  - r: случайные (или симулированные) значения.
- Следующий пример показывает, как можно симулировать данные из нормального распределения, используя функция rnorm.

Пример

```
norm.vals1 <- rnorm(n=10)
norm.vals2 <- rnorm(n=100)
norm.vals3 <- rnorm(n=1000)
norm.vals4 <- rnorm(n=10000)
# set up plotting region
par(mfrow=c(2,2))
hist(norm.vals1,main="10 RVs")
hist(norm.vals2,main="100 RVs")
hist(norm.vals3,main="1000 RVs")
hist(norm.vals4,main="10000 RVs")
```

Гистограммы



Интерпретация результатов

- С ростом размера выборки форма распределения становится больше похожа на нормальное распределение. Про объект norm.vals1 трудно сказать, что он был сгенерирован из нормального распределения с мат.ожиданием 0 и СКО 1. Если посмотреть на суммарную статистику этого объекта, то увидим, что его мат. ожидание и СКО не близки к 0 и 1 соответственно.

```
> c(mean(norm.vals1),sd(norm.vals1))
[1] 0.2461831 0.7978427
```

- Посчитаем МО и СКО объекта norm.vals4, сгенерированного 10,000 случайных значений из распределения N(0, 1):

```
> c(mean(norm.vals4),sd(norm.vals4))
[1] 0.004500385 1.013574485
```

- Для больших симуляций, результат будет еще ближе:

```
> norm.vals5 <- rnorm(n=1000000)
> c(mean(norm.vals5),sd(norm.vals5))
[1] 0.0004690608 0.9994011738
```

Центральная предельная теорема

- При приближении размера n выборки, взятой из популяции с математическим ожиданием μ и дисперсией σ^2, к бесконечности, статистические оценки выборочного распределения будут сходится к рассматриваемым теоретическим распределениям.

Объекты R

Объекты данных в R

- Четыре наиболее часто используемых типа объектов данных в R – это векторы, матрицы, блоки данных и списки.
- Вектор – набор элементов одинакового вида (mode), логических, численных (integer или double), комплексных, символьных или списков.
- Матрица это множество элементов, представленных в виде строк и столбцов, где все элементы одного вида (mode), логических, численных (integer или double), комплексных или символьных.
- Блок данных – то же самое, что и матрица, но колонки могут быть разных видов.
- Список – это обобщение вектора, представляющее собой коллекцию объектов данных.

Создание векторов

- **Функция c**
- Самый простой способ создать вектор – конкатенация с помощью функции c, связывающей вместе символьные, численные или логические элементы.

```
 > value.num <- c(3,4,2,6,20)
> value.char <- c("koala","kangaroo","echidna")
> value.logical.1 <- c(F,F,T,T)
# or
> value.logical.2 <- c(FALSE,FALSE,TRUE,TRUE)
```

- Для логических векторов TRUE и FALSE – логические значения, а T и F – переменные с такими значениями.

Создание векторов

- **Функции rep и seq**
- Функция rep реплицирует элементы векторов. Например,

```
> value <- rep(5,6)
> value
[1] 5 5 5 5 5 5
```

- Функция seq создает регулярную последовательность значений, формирующих вектор.

```
> seq(from=2,to=10,by=2)
[1] 2 4 6 8 10
> seq(from=2,to=10,length=5)
[1] 2 4 6 8 10
> 1:5
[1] 1 2 3 4 5
> seq(along=value)
[1] 1 2 3 4 5 6
```

Создание векторов

- **Комбинирование функций c, rep и seq**

```
> value <- c(1,3,4,rep(3,4),seq(from=1,to=6,by=2))
> value
[1] 1 3 4 3 3 3 3 1 3 5
```

- Элементы вектора должны быть одного вида. Команда

```
> c(1:3,"a","b","c")
```

выдаст сообщение об ошибке.

Создание векторов

- **Функция scan**
- Функция scan используется для ввода данных с клавиатуры.
- Также данные могут считываться из файлов.
- Пример считывания данных с клавиатуры:

```
> value <- scan()
1: 3 4 2 6 20
6:
> value
[1] 3 4 2 6 20
```

Основные вычисления с численными векторами

- Вычисления над векторами производятся поэлементно. При выполнении арифметических операций над векторами, один из которых короче другого, более короткий вектор используется повторно.

```
> x <- runif(10)
> x
[1] 0.3565455 0.8021543 0.6338499 0.9511269
[5] 0.9741948 0.1371202 0.2457823 0.7773790
[9] 0.2524180 0.5636271
> y <- 2*x + 1 # recycling short vectors
> y
[1] 1.713091 2.604309 2.267700 2.902254 2.948390
[6] 1.274240 1.491565 2.554758 1.504836 2.127254
```

Пример

```
> z <- (x-mean(x))/sd(x)
> z
[1] -0.69326707 0.75794573 0.20982940 1.24310440
[5] 1.31822981 -1.40786896 -1.05398941 0.67726018
[9] -1.03237897 -0.01886511
> mean(z)
[1] -1.488393e-16
> sd(z)
[1] 1
```

Функции, которые дают результат такой же длины

Function	Description
cos, sin, tan	Cosine, Sine, Tangent
acos, asin, atan	Inverse functions
cosh, sinh, tanh	Hyperbolic functions
acosh, asinh, atanh	Inverse hyperbolic functions
log	Logarithm (any base, default is natural logarithm)
log10	Logarithm (base 10)
exp	Exponential (e raised to a power)
round	Rounding
abs	Absolute value
ceiling, floor, trunc	Truncating to integer values
gamma	Gamma function
lgamma	Log of gamma function
sqrt	Square root

Функции, результатом которых является число

Function	Description
sum	Sum elements of a vector
mean	arithmetic mean
max, min	Maximum and minimum
prod	Product of elements of a vector
sd	standard deviation
var	variance
median	50th percentile

Создание матриц

- **Функции dim и matrix**
- Функция dim может использоваться для конвертации вектора в матрицу

```
> value <- rnorm(6)
> dim(value) <- c(2,3)
> value
[,1] [,2] [,3]
[1,] 0.7093460 -0.8643547 -0.1093764
[2,] -0.3461981 -1.7348805 1.8176161
```

- Чтобы конвертировать назад в вектор, надо опять применить функцию dim.

```
dim(value) <- NULL
```

- Или можно использовать функцию matrix

```
> matrix(value,2,3)
[,1] [,2] [,3]
[1,] 0.7093460 -0.8643547 -0.1093764
[2,] -0.3461981 -1.7348805 1.8176161
```

- Если хотим заполнять по строкам

```
> matrix(value,2,3,byrow=T)
[,1] [,2] [,3]
[1,] 0.709346 -0.3461981 -0.8643547
[2,] -1.734881 -0.1093764 1.8176161
```

Создание матриц

- **Функции rbind и cbind**
- Чтобы привязать строку к уже существующей матрице, используется функция rbind

```
> value <- matrix(rnorm(6),2,3,byrow=T)
> value2 <- rbind(value,c(1,1,2))
> value2
[,1] [,2] [,3]
[1,] 0.5037181 0.2142138 0.3245778
[2,] -0.3206511 -0.4632307 0.2654400
[3,] 1.0000000 1.0000000 2.0000000
```

- Чтобы привязать столбец к уже существующей матрице, используется функция cbind

```
> value3 <- cbind(value2,c(1,1,2))
[,1] [,2] [,3] [,4]
[1,] 0.5037181 0.2142138 0.3245778 1
[2,] -0.3206511 -0.4632307 0.2654400 1
[3,] 1.0000000 1.0000000 2.0000000 2
```

Функция data.frame

- Функция data.frame конвертирует матрицу или коллекцию векторов в блок данных

```
> value3 <- data.frame(value3)
> value3
X1 X2 X3 X4
1 0.5037181 0.2142138 0.3245778 1
2 -0.3206511 -0.4632307 0.2654400 1
3 1.0000000 1.0000000 2.0000000 2
```

- В другом примере соединим две колонки данных вместе.

```
> value4 <- data.frame(rnorm(3),runif(3))
> value4
rnorm.3. runif.3.
1 -0.6786953 0.8105632
2 -1.4916136 0.6675202
3 0.4686428 0.6593426
```

Блоки данных

- Имена строк и столбцов в блоке данных создаются по умолчанию, но их можно поменять, используя функции names и row.names. Посмотрим названия строк и столбцов блока данных:

```
> names(value3)
[1] "X1" "X2" "X3" "X4"
> row.names(value3)
[1] "1" "2" "3"
```

- Другие метки можно присвоить следующим образом:

```
> names(value3) <- c("C1","C2","C3","C4")
> row.names(value3) <- c("R1","R2","R3")
```

- Также можно определять имена с помощью самой функции data.frame.

```
> data.frame(C1=rnorm(3),C2=runif(3),row.names=c("R1","R2","R3")
C1 C2
R1 -0.2177390 0.8652764
R2 0.4142899 0.2224165
R3 1.8229383 0.5382999
```

Доступ к элементам векторов и матриц через индексирование

- Индексирование может осуществляться через
  - Вектор положительных чисел, чтобы указывать включение
  - Вектор отрицательных чисел, чтобы указывать включение
  - Вектор логических значений, чтобы указывать, какие элементы нужны, а какие нет
  - Вектор имен, если у объекта есть атрибут names
- В последнем случае, если справа стоит нулевой индекс, элементы не выбираются. Если нулевой индекс появляется слева, не происходит присваивание.

Индексирование векторов

- Создаем случайный набор значений от 1 до 5 из 20 элементов, определяем, какие элементы равны 1.

```
> x <- sample(1:5, 20, rep=T)
> x
[1] 3 4 1 1 2 1 4 2 1 1 5 3 1 1 1 2 4 5 5 3
> x == 1
[1] FALSE FALSE TRUE TRUE FALSE TRUE FALSE FALSE TRUE
[10] TRUE FALSE FALSE TRUE TRUE TRUE FALSE FALSE FALSE
[19] FALSE FALSE
ones <- (x == 1) # parentheses unnecessary
```

- Заменим единицы нулями и сохраним значения большие 1 в объект y.

```
> x[ones] <- 0
> x
[1] 3 4 0 0 2 0 4 2 0 0 5 3 0 0 0 2 4 5 5 3
> others <- (x > 1) # parentheses unnecessary
> y <- x[others]
> y
[1] 3 4 2 4 2 5 3 2 4 5 5 3
```

- Следующая команда возвращает позиции элементов вектора x, больших 1
- that is greater than 1.

```
> which(x > 1)
[1] 1 2 5 7 8 11 12 16 17 18 19 20
```

Индексирование блоков данных

- Блоки данных индексируются или через строки и столбцы с использованием специального имени, которое соответствует строке или столбцу, или с использованием номеров. below.
- Индексирование по столбцу:

```
> value3
C1 C2 C3 C4
R1 0.5037181 0.2142138 0.3245778 1
R2 -0.3206511 -0.4632307 0.2654400 1
R3 1.0000000 1.0000000 2.0000000 2
> value3[, "C1"] <- 0
> value3
C1 C2 C3 C4
R1 0 0.2142138 0.3245778 1
R2 0 -0.4632307 0.2654400 1
R3 0 1.0000000 2.0000000 2
```

Индексирование блоков данных

- Индексирование по строке:

```
> value3["R1", ] <- 0
> value3
C1 C2 C3 C4
R1 0 0.0000000 0.0000000 0
R2 0 -0.4632307 0.2654400 1
R3 0 1.0000000 2.0000000 2
> value3[] <- 1:12
> value3
C1 C2 C3 C4
R1 1 4 7 10
R2 2 5 8 11
R3 3 6 9 12
```

Индексирование блоков данных

- Чтобы получить доступ к первым двум строкам матрицы или блока данных:

```
> value3[1:2,]
C1 C2 C3 C4
R1 1 4 7 10
R2 2 5 8 11
```

- Чтобы получить доступ к первым двум столбцам матрицы или блока данных:

```
> value3[,1:2]
C1 C2
R1 1 4
R2 2 5
R3 3 6
```

- Чтобы получить доступ к элементам со значением больше 5:

```
> as.vector(value3[value3>5])
[1] 6 7 8 9 10 11 12
```

Создание списков

- Списки создаются с помощью функции list. Могут включать элементы различных видов, длины и размера

```
> L1 <- list(x = sample(1:5, 20, rep=T),
y = rep(letters[1:5], 4), z = rpois(20, 1))
> L1
$x
[1] 2 1 1 4 5 3 4 5 5 3 3 3 4 3 2 3 3 2 3 1
$y
[1] "a" "b" "c" "d" "e" "a" "b" "c" "d" "e" "a" "b“
[13] "c" "d" "e" "a" "b" "c" "d" "e"
$z
[1] 1 3 0 0 3 1 3 1 0 1 2 2 0 3 1 1 0 1 2 0
```

Доступ к элементам списка

- Через имя или по номеру позиции, на которой находится элемент, с использованием операции взятия подмножества [[]] или извлечения $.

```
> L1[["x"]]
[1] 2 1 1 4 5 3 4 5 5 3 3 3 4 3 2 3 3 2 3 1
> L1$x
[1] 2 1 1 4 5 3 4 5 5 3 3 3 4 3 2 3 3 2 3 1
> L1[[1]]
[1] 2 1 1 4 5 3 4 5 5 3 3 3 4 3 2 3 3 2 3 1
```

- Чтобы извлечь подсписок, используются одинарные скобки:

```
> L1[1]
$x
[1] 2 1 1 4 5 3 4 5 5 3 3 3 4 3 2 3 3 2 3 1
```