Методические указания к лабораторным работам по курсу «Методы анализа данных»

Меретилов М.А., КГТУ

Красноярск, 2006

Аннотация

Курс «Методы анализа данных» имеет своей целью ознакомить студентов с методами и подходами анализа и обработки данных, встречающихся в инженерных задачах. Основанный на теории вероятностей и математической статистике, он неразрывно связан с прикладной стороной инженерной деятельности — производством и реализацией алгоритмов анализа, представлением результатов решения поставленных задач.

Специалисты данной и смежных областей должны владеть специальными инструментами и средствами, позволяющими решать поставленные задачи самым эффективным образом. Владение такими средствами и навыками алгоритмизации позволяет молодым специалистам решать возникающие на производстве задачи, задумываясь в основном над путём решения, а не над конкретными вопросами реализации. Не секрет, что одними из основных проблем, встающих на пути автоматизации производства и оказания услуг, являются устаревшие средства и недостаточное понимание сути происходящих явлений, в том числе и приложение больших усилий там, где можно обойтись малыми. В случаях, когда значительный выигрыш могут дать специализированные средства, используются языки программирования общего назначения и часто низкоуровневые. Неудивительно, что большая часть времени инженера уходит на разработку и отладку таких средств, и на тестирование не остаётся времени.

В предлагаемых методических указаниях используется система статистических расчётов R[1], прародительница которой, система S, появилась ещё в 1976 году. Несмотря на свой солидный возраст, эта система поддерживается и развивается, на официальном сайте есть репозиторий[2] пакетов для разных целей, который регулярно пополняется исследователями-энтузиастами. Система R является свободной системой, распространяемой под лицензией GNU GPL версии 2 и поддерживает платформы Windows (32-х битную), UNIX (в том числе Linux) и MacOS X. С основного сайта проекта можно загрузить исходные тексты системы и пакетов, а также бинарные сборки для некоторых из поддерживаемых платформ.

Солидный для систем подобного рода возраст R идёт ей только на пользу — это «выстоявшийся» продукт с устойчивым и документированным поведением, с давно исправленными ошибками, свободно распространяемый и расширяемый. Именно эти качества, наряду с простотой, функциональностью и ориентированностью на статистический анализ и исследования, послужили причиной выбора системы R в качестве основы для практических занятий.

Здесь и далее наряду с выражением «система R» будет употребляться и просто «R» в качестве обозначения этой системы.

[1] `http://www.r-project.org`, последняя версия на момент написания была 2.2.1

[2] CRAN, Comprehensive R Archive Network, `http://cran.r-project.org`

1 Основы работы с R

В данном разделе будут даны основы работы с системой R, остальную информацию всегда можно получить из хорошо составленной документации R, справки по функциям и описания пакетов. Здесь даётся лишь информация, необходимая для выполнения лабораторных работ и позволяющая ориентироваться в системе.

Команды, введённые в командной строке, будут обозначаться машинописным шрифтом. В случае приведения нескольких строк (возможно, вывод команды) приглашение ввода команды будет обозначаться символом >:

```
> 7
[1] 7
> q()
```

В случае, если команда «растягивается» на несколько строк, приглашение в каждой последующей строке будет обозначаться знаком +.

Некоторую строку символов можно сделать комментарием, начав её со знака #.

1.1 Начало работы с R: числа, векторы, переменные, списки

1.1.1 Работа с командной строкой

Основная рабочая среда системы R — командная строка. Поэтому при запуске системы вы увидите краткую информацию о работе с командной строкой и приглашение ввода. Вы всегда можете выйти, введя команду `q()`. На самом деле, эта команда является вызовом функции с именем «q» и без аргументов. Эта функция производит выход из системы.

Для того, чтобы получить справку о некоторой функции, нужно вызвать функцию `help(<имя-функции>)` с этим именем в качестве аргумента, в кавычках или без (например, `help("help")` или `help(help)`). Аналогом этой функции является команда `?<имя-функции>`, но в этом случае имя функции приводится без кавычек (например, `?help`). Оба этих способа приведут к одному и тому же — в случае наличия справки по этой функции будет открыто новое окно (создана новая терминальная сессия, если используется версия для UNIX) с содержимым этой справки. Выйти из просмотра справки можно, нажав клавишу «q».

Последовательность команд R можно вводить в файл с последующей загрузкой этого файла как цельного сценария. Такая загрузка осуществляется функцией `source(<имя-файла>)`.

1.1.2 Основные типы данных в R

Основным типом данных в системе R является число. Числа представляются обычным для языков программирования образом — арабские цифры, десятичная дробь разделяется точкой, маленькие и большие числа можно обозначать в виде `-0.4+E3`. При вводе в качестве команды числа, в результате будет выведено оно же.

Вектор — это набор нескольких чисел. Количество чисел называется *длиной вектора*. Формируется вектор функцией `c(<число1>, <число2>, ...)`, если ввести её после приглашения R, то выведется значение вектора:

```
> c(1, 2, 3, 4, 5)
[1] 1 2 3 4 5
```

Такой же вектор, состоящий из последовательных чисел, можно получить и командой `<начальное-число>:<конечное-число>` (например, `1:5`). Такая команда позволяет получать вектор из последовательных целых чисел от начального до конечного включительно.

Вектор из последовательных нецелых чисел можно получить, воспользовавшись функцией `seq(<начальное-число>, <конечное-число>, seq=<количество-чисел-в-векторе>)`, которая создаёт вектор заданной длины, элементами которого являются равномерно распределённые числа из интервала [`<нач-число>`; `<кон-число>`]:

```
> seq(1, 3, seq=3)
[1]   1   1.5  3
```

Отдельно от всех чисел стоят псевдо-числа `NaN` (Not a number) и `NA` (Not assigned). Первое является результатом выражений-неопределённостей ($0/0$, $\infty - \infty$), а второе используется как синоним не определённого пока значения (аналог NULL в системах управления базами данных).

1.1.3 Переменные в R

В языке системы R есть возможность хранить данные в переменных. Переменные могут иметь имена, состоящие из символов латинского алфавита, арабских цифр, символов подчёркивания и точки. Имена могут начинаться только с алфавитного символа или символа точки, но в последнем случае после точки может идти только алфавитный символ.

Имена переменных чувствительны к регистру.

Присваивание переменных осуществляется двумя способами — привычным многим программистам знаком `=` и не таким привычным в данном качестве знаком «стрелка» `<-`:

```
> x = 7
> x
[1] 7
> x <- 8
> x
[1] 8
```

В данном случае команда, состоящая просто из имени переменной, выводит значение этой переменной. В системе R подобным образом легко вывести значения переменных всех типов, и это позволяет в любое время отслеживать такие значения, не прибегая к серьёзным методам отладки.

Векторам можно присваивать не только числа, но и, например, векторы (`x = c(1, 2, 3)`) и результаты выражений (`x = c(1, 2, 3) + c(2, 3, 4)`). Об операциях над векторами будет сказано в следующем подразделе.

Доступ к элементам вектора можно получить, указав номер этого элемента в квадратных скобках после вектора (переменной, содержащей вектор, или выражения, результатом которого будет вектор):

```
> x = 1:20        # 1-й случай
> x[5]
[1] 5
> (1:20)[5]       # 2-й случай
[1] 5
> 1:20[5]         # 3-й случай
Ошибка в 1:20[5] : NA/NaN-аргумент
```

В третьем случае система считала, что индекс 5 применяется к числу 20, что и вызвало ошибку. Правильный пример указан во втором случае, выражение 1:20 надо взять в скобки.

Также можно более гибко адресовать элементы вектора при использовании квадратных скобок. Можно выбрать сразу несколько интересующих элементов, указав в квадратных скобках вектор, содержащий индексы необходимых элементов:

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

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

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

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

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

При присваивании переменной значения, если этой переменной не было, она создаётся. В результате может быть создано много переменных, не все из которых важны. Просмотреть список созданных переменных можно, вызвав функцию `ls()`. Если некоторые из переменных уже не нужны, их можно удалить функцией `rm(<переменные>)`. Это удобно, если значения переменных нужно сохранять между сессиями работы, и лишние переменные будут только мешать.

1.1.4 Операции над основными типами данных

Над числами и векторами можно производить обычные арифметические операции, и обозначаются они довольно традиционно:

- сложение: `2 + 3`;

- вычитание: `1 - 2`;
- умножение: `5 * 2`;
- деление: `5 / 2`;
- возведение в степень: `2 ^ 8`.

В случае проведения операции над векторами одинаковой размерности, эта операция производится почленно над каждой парой элементов векторов-участников, и результатом является вектор, состоящий из результатов этого действия. В случае, когда размерность векторов не совпадает, производится приведение длины более «короткого» вектора к длине «длинного» вектора. Приведение выполняется следующим образом: вектор приобретает длину, такую же, как и у более «длинного», и содержимое этих новых элементов берётся из содержимого старых элементов вектора, повторяемых циклически:

```
> c(1, 2, 3) + c(2, 2, 2, 2, 2, 2, 2)
[1] 3 4 5 3 4 5 3
> 1 + c(1, 1)
[1]  2  2
```

Во втором случае число `1` представляется как вектор единичной длины, и он приводится к более длинному вектору – участнику выражения.

В выражениях могут участвовать привычные глазу C-программиста функции `sin(x)`, `cos(x)`, `tan(x)`, `sqrt(x)` и некоторые другие. Их аргументами могут быть как числа, так и векторы.

Отдельное место в языке R занимают встроенные функции работы с векторами. К таким функциям относятся:

`sum(v)`: нахождение суммы элементов вектора `v`;

`max(v)`: нахождение максимального элемента вектора `v`;

`min(v)`: нахождение минимального элемента вектора `v`;

`prod(v)`: нахождение произведения элементов вектора `v`;

`length(v)`: нахождение количества элементов вектора `v`;

`mean(v)`: нахождение среднего (оценки математического ожидания) вектора `v`;

`var(v)`: нахождение вариации (оценки дисперсии) вектора `v`;

`sort(v)`: возвращение вектора с такой же длиной, как и у `v`, элементы которого отсортированы в порядке возрастания.

При помощи этих функций операции с векторами принимают вид простых арифметических выражений, в отличие от громоздких циклов, выполняющих те же функции, в языках программирования общего назначения. Так, функцию `mean(v)` можно заменить выражением `sum(v)/length(v)`. В результате смысл этих выражений быстро становится ясен даже при беглом взгляде, и они становятся несложными в отладке.

1.1.5 Списки

1.2 Визуализация двумерных графиков

1.2.1 Функции рисования

Для того, чтобы нарисовать график какой-нибудь зависимости, используется функция `plot()`. В неё передаются как минимум два аргумента — вектор значений абсцисс, и вектор значений ординат:

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

Кроме этих двух аргументов, эта функция принимает ещё много аргументов, отвечающих за оформление и некоорые другие аспекты графического вывода, о некоторых из них подробнее будет сказано в следующем разделе, о других — в документации к системе.

Функция `plot()` не только рисует график, но ещё и создаёт (а в случае наличия уже созданного обнуляет) новое «графическое устройство»[3]. Нам достаточно рассматривать «графическое устройство» как окно, в котором рисуется график: есть окно — можно туда дорисовывать элементы графика или дополнительные графики, нет — надо его создать функцией `plot()`, которая нарисует основной график.

К нарисованному функцией `plot()` графику можно добавлять другие при помощи той же функции `plot()` со специальным аргументом, о котором будет сказано в следующем подразделе.

1.2.2 Дополнительные аргументы функций рисования

Помимо аргументов-векторов абсцисс и ординат в функцию `plot()` могут передаваться следующие аргументы:

- `add=TRUE` обозначает режим добавления графика вместо того, чтобы создать новое «графическое устройство» или очистить имеющееся;
- `log="x"` или `log="y"` или `log="xy"` делает ось абсцисс, ординат или обе оси логарифмическими;
- `type=<тип-графика>` позволяет указать тип выводимого графика; основные — `p` (точки, по умолчанию), `l` (линии), `b` (линии с точками) и `n` (ничего не рисуется);
- `xlab=<название-оси-абсцисс>` и `ylab=<название-оси-ординат>` позволяет указать название для осей абсцисс и ординат соответственно;
- `main=<основная-надпись-графиков>` и `sub=<подпись-маленькими-буквами>` создают надписи на графике.

Например, чтобы задать надписи для координатных осей, можно использовать следующий вариант:

```
> plot(x, y, xlab="x", ylab="Синус x")
```

[3] В качестве графических устройств могут выступать как окна на экране, так и файлы GIF, JPEG, PNG, PostScript и PDF. Подробнее об этом можно почитать во «Введении в R (An Introduction to R)» на английском языке из документации R (файл `R-intro.pdf`)

1.3 Многомерные данные: массивы и матрицы

В математике часто используют матричную запись для, например, систем алгебраических уравнений — такая запись лаконична, её всегда можно понять. В R есть поддержка матриц и данных с более чем двумя измерениями. Соответственно, в конструкциях на языке R можно использовать такую же лаконичную запись для громоздких выражений, как и аналитическая запись на бумаге.

1.3.1 Доступ к многомерным данным

В R есть один тип многомерных данных — массив. Матрица является частным случаем массива, тогда, когда этот массив имеет два измерения.

Функция `diag(<вектор>)` возвращает диагональную матрицу, значениями диагонали которой являются соответствующие элементы вектора.

Массив создаётся функцией `array(<вектор-данных>, <вектор-измерений>)`. `<Вектор-данных>` вредставляет собой вектор чисел, которые станут содержимым массива. `<Вектор-измерений>` представляет из себя вектор чисел, количество которых задаёт количество измерений массива, а значение каждого из чисел — размерность массива в соответствующем измерении. Например:

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

Как мы видим, содержимым для массива мы выбрали числа от 1 до 20. Распределение этих чисел по массиву происходит следующим образом: первый элемент вектора становится элементом массива с индексами $(1, 1)$, следующий — элементом с индексами $(2, 1)$ и так далее до тех пор, пока мы не заполним весь первый индекс массива (в примере выше это делалось 4 раза — именно такую размерность имело первое измерение). Потом таким же образом заполняются элементы с индексами $(1, 2) \ldots (4, 2)$, второй индекс опять увеличивался на единицу и так далее.

Таким же образом заполняется всякий многомерный массив. Существует правило, которое легко запомнить — «Чаще всего меняется первый индекс, и реже всего — последний».

Теперь получить конкретный элемент массива мы можем, зная его индексы. Они указываются, как и в случае векторов, в квадратных скобках после имени переменной, содержащей массив. Индексы перечисляются через запятую. То есть, захотев получить элемент массива из примера выше со значением 10, нужно использовать следующую конструкцию: `x[2, 3]`. Индексы мы можем узнать из вывода значения всего массива, приведённого в том же примере.

Надо обратить внимание, что вывод массива на экран выполняется срезами (в случае матрицы этот срез один, но для многомерного массива вывод будет осуществлён последовательно, срез за срезом). Например:

```
> x = array(1:100, c(5, 10, 2))
```

```
> x
, , 1

     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    1    6   11   16   21   26   31   36   41    46
[2,]    2    7   12   17   22   27   32   37   42    47
[3,]    3    8   13   18   23   28   33   38   43    48
[4,]    4    9   14   19   24   29   34   39   44    49
[5,]    5   10   15   20   25   30   35   40   45    50

, , 2

     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]   51   56   61   66   71   76   81   86   91    96
[2,]   52   57   62   67   72   77   82   87   92    97
[3,]   53   58   63   68   73   78   83   88   93    98
[4,]   54   59   64   69   74   79   84   89   94    99
[5,]   55   60   65   70   75   80   85   90   95   100
```

Как видно, срезы этого трёхмерного массива представляются индексами вида `, , x`, где `x` — это номер среза (и заодно значение третьего индекса).

1.3.2 Операции над многомерными данными

Простейшие операции над матрицами и массивами — обычные арифметические операции над массивамии числами или массивами такой же размерности, результатом которых будут массивы той же размерности, над соответствующими элементами которых выполнена указанная операция:

```
> array(1:20, c(4, 5)) + 1
     [,1] [,2] [,3] [,4] [,5]
[1,]    2    6   10   14   18
[2,]    3    7   11   15   19
[3,]    4    8   12   16   20
[4,]    5    9   13   17   21
> array(1:20, c(4, 5))^2
     [,1] [,2] [,3] [,4] [,5]
[1,]    1   25   81  169  289
[2,]    4   36  100  196  324
[3,]    9   49  121  225  361
[4,]   16   64  144  256  400
> array(1:20, c(4, 5)) + array(1:20, c(4, 5))
     [,1] [,2] [,3] [,4] [,5]
[1,]    2   10   18   26   34
[2,]    4   12   20   28   36
[3,]    6   14   22   30   38
[4,]    8   16   24   32   40
> array(1:20, c(4, 5)) + array(1:20, c(5, 4))
```

```
Ошибка в array(1:20, c(4, 5)) + array(1:20, c(5, 4)) :
        неподобные многомерные матрицы
```

Матричное умножение двух матриц выполняется с помощью операции `%*%`:

```
> array(1:20, c(4, 5)) %*% array(1:20, c(5, 4))
     [,1] [,2] [,3] [,4]
[1,]  175  400  625  850
[2,]  190  440  690  940
[3,]  205  480  755 1030
[4,]  220  520  820 1120
```

Транспонирование матрицы можно выполнить функцией `t(<матрица>)`.

Функция `crossprod(X, y)` выполняет то же самое, что и выражение `t(X) %*% y`, но делает это быстрее (так как используется реализация этой операции на языке низкого уровня, которая быстрее выполняется).

Нахождение матрицы, обратной данной — неотъемлемая часть решения системы линейных алгебраических уравнений

$$\mathbf{A} \cdot \mathbf{x} = \mathbf{b}, \text{ где}$$

$$\mathbf{x} = \mathbf{A}^{-1} \cdot \mathbf{b}.$$

И в R этим занимается функция `solve(A, b)`, возвращающая вектор `x`, являющийся решением вышеприведённой системы. Употребление функции `solve(A)` без второго аргумента возвращает матрицу, обратную `A` ($\mathbf{A}^{-1}$).

1.4 Возможности языка R: условия, циклы, функции

Язык R содержит в себе черты алгоритмических языков программирования, и добавляет к ним лаконичность работы с математической составляющей. В нём присутствуют такие черты, как комментарии, блоки выражений, проверка условий, циклическое выполнение выражений и задание функций с последующим их вызовом.

1.4.1 Блоки выражений

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

```
> {
+ x = 7;
+ y = 8;
+ x + y;
+ }
[1] 15
```

1.4.2 Проверка условий

Проверка условий осуществляется неотличимо от языка C:

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

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

1.4.3 Циклы

Циклические выражения с заданным множеством итераций выполняются при помощи конструкции

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

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

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

В случае, когда нужно итерировать некоторое выражение до тех пор, пока выполняется какое-либо условие, используется цикл

```
while (<условие>)  <выражение>
```

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

1.4.4 Объявление и вызов функций

Функция в R — это объект, принимающий аргументы и возвращающий некоторое значение. Сам по себе этот объект существовать не может, и обычно он присваивается переменной. Соответственно, объявление функции является присваиванием этой функции некоторой переменной, через которую можно будет впоследствии обращаться к функции. Общий вид такого объявления приведён ниже:

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

Здесь обращение к функции осуществляется через переменную, этой функции можно передать аргументы и результатом вызова функции будет значение выражения.

Дальнейший вызов будет осуществляться в таком виде:

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

Тогда в переменную результат будет помещено возвращаемое значение данной функции.

Аргументы могут задаваться со значениями по умолчанию:

```
func = function(x, y, distribution="norm") <...>
```

и вызовы этой функции могут иметь вид:

```
func(1, 7, "exp")
func(1, 8)
```

В первом случае, как мы видим, указаны все аргументы функции непосредственно, а во втором — указаны только аргументы, у которых нет значений по умолчанию, не указанный здесь distribution будет иметь значение "norm".

Можно указывать аргументы в любом порядке, но тогда нужно явно определять, какой именно аргумент имеется в виду. Для этого используется уже известный нам знак =. Это вторая его роль в R:

```
func(distribution="norm", x=2, y=7)
func(dist="exp", y=8, x=1)
```

Как показано в последнем случае, именовать аргумент можно не полностью, а только первыми буквами, лишь бы исключить двоякое толкование этого сокращения. Таким образом, вы можете не беспокоиться о длинном названии аргумента функции, если начальная часть этого названия может использоваться с той же информативностью (например, dist — частое сокращение слова distribution в программировании).

В качестве выражения – тела функции может выступать либо одиночное выражение, либо блочное. Во втором случае возвращаемым значением функции будет значение последнего выражения в блоке.

Если в рабочей среде R уже инициализированы переменные с именами, овпадающими с аргументами функции, аргументы их заместят — у них более преимущественная область видимости переменных.

Все присваивания значений переменным, которые осуществляются внутри выражения – тела функции, являются внутренними для этой функции, и эти изменения не сохраняются при выходе из неё. Для того, чтобы глобально присвоить значение переменной внутри функции, используется конструкция <<-. Следующий пример поясняет ситуацию:

```
> func <- function(val)
+ {
+     x = val;
+     y <- val;
+     z <<- val;
+ }
> x = 5
```

```
> y = 6
> z = 7
> x; y; z
[1] 5
[1] 6
[1] 7
> val = 17
> func(11)
> x; y; z
[1] 5
[1] 6
[1] 11
```

Во-первых, присваивание внутри функции повлияло только на переменную `z`, так как использовался специальный оператор присваивания `<<-`, а остальные операторы действуют только внутри функции. Во-вторых, предварительное объявление и присваивание значения переменной `val` не оказало никакого влияния на работу функции — внутри неё эта переменная заменилась одноимённым аргументом.

Кроме всего прочего, в приведённом примере мы видим функцию, которая вроде не возвращает никакого значения. Но на самом деле, если результат функции присвоить какой-нибудь переменной, и вывести её на экран, мы увидим, что это не так:

```
> xx <- func(11)
> xx
[1] 11
```

Дело в том, что неявным результатом последнего выражения стал результат выражения справа от знака присваивания — он и послужил возвращаемым значением.

1.5 Распределения вероятностей

1.5.1 Работа со значениями случайных величин

Так как R создавался специально для решения задач статистического анализа данных, в нём есть замечательная библиотека `stats`, которая подключена по умолчанию.

Кроме многого другого, она включает в себя таблицы многих распространённых распределений вероятности:

Название	Название в R	Аргументы
бета-распределение	`beta`	`shape1`, `shape2`, `ncp`
биномиальное	`binom`	`size`, `prob`
Вейбулла	`weibull`	`shape`, `scale`
гамма-распределение	`gamma`	`shape`, `scale`
лог-нормальное	`lnorm`	`meanlog`, `sdlog`,
нормальное	`norm`	`mean`, `sd`
Пуассоновское	`pois`	`lambda`
равномерное	`unif`	`min`, `max`
распределение Стьюдента	`t`	`df`, `ncp`

распределение Фишера	`f`	`df1`, `df2`, `ncp`
хи-квадрат	`chisq`	`df`, `ncp`
экспоненциальное	`exp`	`rate`

Работа с распределениями осуществляется, опираясь на из названия в R. У таблиц распределений существуют по четыре функции на каждое распределение, их имена составлены из одной буквы, обозначающей предназначение функции, и названия распределения в R. Буква, обозначающая предназначение функции, может быть одной из:

- `d` density, плотность распределения в зависимости от квантиля (если в качестве квантиля передать вектор, то будет возвращён вектор плотностей);
- `p` probability, значение функции распределения в зависимости от квантиля;
- `q` quantile, значение вероятности в зависимости от квантиля;
- `r` random, нахождение вектора случайных чисел, распределённых по соответствующему закону; размер вектора передаётся первым аргументом.

```
> rnorm(5)
[1]  0.1769307  0.8181023 -0.1208060  0.2846609 -0.4915929
> dnorm(0) == 1/sqrt(2*pi)
[1] TRUE
```

В первой строке выводится 5 случайных чисел, распределённых по закону N[0; 1]. Во второй строке выводится результат логического выражения — проверка на равенство значения плотности нормально распределённой случайной величины N[0; 1]. Как и ожидалось, это значение равно $1/\sqrt{2 \cdot \pi}$.

1.5.2 Визуализация случайных величин

Имея некоторую произвольную выборку, можно визуально оценить её, обратив внимание на закон распределения и другие качества.

Основную информацию о выборке можно получить, используя функцию `summary(<вектор>)`:

```
> v = rnorm(100)
> summary(v)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-2.37300 -0.74150  0.12440  0.01619  0.61960  2.58500
```

Эта функция сообщает минимальное и максимальное значения, медиану, среднее и первый и третий квартиль. По тому, что соответствующие значения примерно симметричны относительно среднего (математического ожидания), можно судить, что закон распределения выборки симметричен.

Для вывода гистограммы произвольной выборки используется функция `hist(<вектор>)`:

```
> hist(v)
```

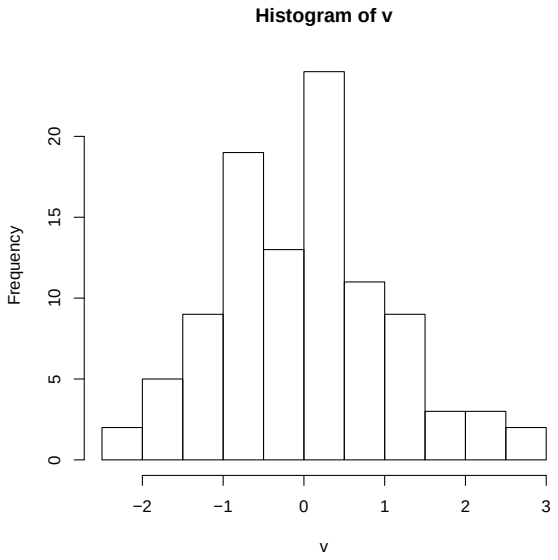

Рис. 1: Результат выполнения функции `hist()`

Эта команда выведет график, наподобие указанного на рис. 1

Рассчитать непараметрическую оценку плотности распределения для произвольной выборки можно при помощи функции `density(<вектор>)`. Вывести эту оценку можно, передав результат функции `plot()`. Например, для нашего примера оценка будет выглядеть как на рис. 2. Этого мы добились, используя команду `plot(density(v))`.

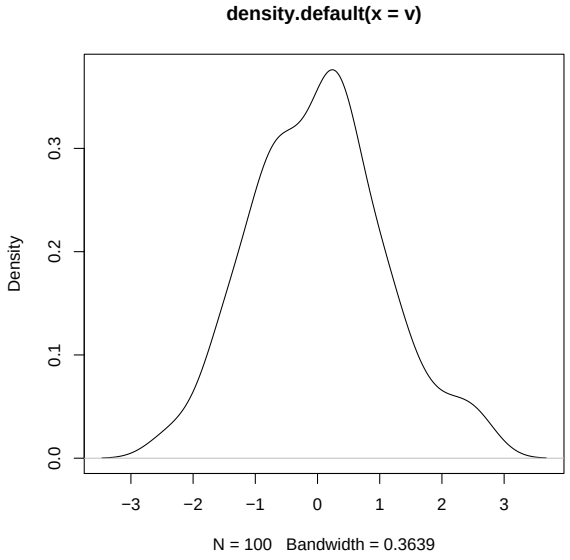

Рис. 2: Результат выполнения функции `density()`

Также для визуальной оценки близости распределения какому-то известному используются графики квантиль-квантиль (Q-Q). Например, чтобы сравнить нашу выборку

с нормально распределённой, можно использовать функцию `qqnorm(<вектор>)`, дополненную функцией `qqline(<вектор>)`. Первая из них выводит график квантиль-квантиль для вектора и нормально распределённой случайной величины. Если график представляет собой точки, находящиеся на диагонали, то распределения идентичны. Чтобы различие было нагляднее, используется функция `qqline()`, она выводит эту самую идеальную диагональ. В итоге, результат будет выглядеть так, как показано на рис. 3.

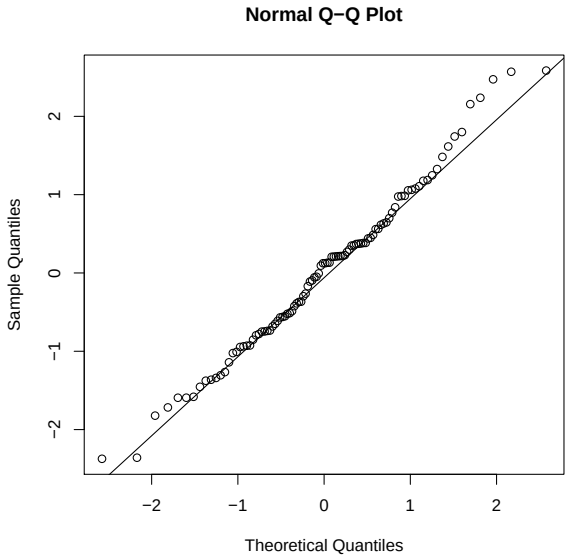

Рис. 3: Результат выполнения функций `qqnorm()` и `qqline()`

В библиотеке `stats` существует ещё много полезных функций, но их описание выходит за рамки этого пособия.

1.6 Формирование выражений

2 Лабораторные работы