КАЗАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет географии и экологии

Основные понятия языка R

Учебно-методическое пособие

КАЗАНЬ - 2007

Составители:
доктор биологических наук, доцент А.А.Савельев,
старший преподаватель С.С.Мухарамова,
старший преподаватель А.Г.Пилюгин
Е.А.Алексеева

Учебно-методическое пособие предназначено для студентов естественных факультетов, изучающих курс «Теория вераятности и математическая статистика». Даются основные понятия языка R, разбираются примеры использования операторов, методы анализа и обработки. предназначенной для выполнения практических заданий по курсам «ГЕОСТАТИСТИКА» и «Теория вераятности и математическая статистика». Печатается по решению учебно-методической комиссии факультета географии и экологии

Введение

Цель этого пособия состоит в том, чтобы дать отправную точку для людей, недавно заинтересовавшихся системой R.

R – статистическая система анализа, созданная Россом Ихакой и Робертом Гентлеманом (1996, *J.Comput. Граф. Stat.,* 5: 299-314). R является и языком и программным обеспечением; его наиболее замечательный особенности:

- эффективная обработка данных и простые средства для сохранения результатов,
- набор операторов для обработки массивов, матриц, и других сложных конструкций,
- большая, последовательная, интегрированная коллекция инструментальных средств для проведения статистического анализа,
- многочисленные графические средства,
- простой и эффективный язык программирования, который включает много возможностей.

Язык **R** - рассматривают как диалект языка **S** созданный AT&T Бэлл Лаборатории. **S** доступен как программное обеспечение S-PLUS коммерческой системы MathSoft (см.http://www.splus.mathsoft.com для получения дополнительной информации). Есть существенные различия в концепции **R** и **S** (те, кто хочет знать больше об этом может читать статью, написанную Gentleman и Ihaka (1996) или R-FAQ (часто задаваемые вопросы) (http://cran.r-project.org/doc/FAQ/R-FAQ.html).

R доступен в нескольких формах: исходный текст программ, написанный на C (и некоторые подпрограммы в Fortran77) и в откомпилированном виде.

R – язык со многими функциями для выполнения статистического анализа и графического отображения результатов, которые визуализируется сразу же в собственном окне и могут быть сохранены в различных форматах (например, jpg, png, bmp, eps, или wmf под Windows, ps, bmp, pictex под Unix).

Результаты статистического анализа могут быть отображены на экране. Некоторые промежуточные результаты (*P*-values, коэффициент регрессии и т.п.) могут быть сохранены в файле и использоваться для последующего анализа.

R – язык, позволяющий пользователю использовать операторы циклов, чтобы последовательно анализировать несколько наборов данных. Также возможно объединить в отдельную программу различные статистические функции, для проведения более сложного анализа.

Сначала, R может показаться слишком сложным для неспециалиста (например, биолога). На самом деле это не так. Главная особенность R – ее гибкость. Например, анализ может быть сделан без отображения результата. Действительно, иногда необходима только часть результатов, которая представляет интерес.

1. Основные операторы

Инсталлированная система R запускается вызовом файла (RGui.exe или Rterm.exe R под Unix). Появившееся подсказка ‘>’ указывает, что R ждет ваших команд. Под Windows, некоторые команды, связанные с операционной системой (доступ к справке, открытию файлов...) могут быть выполнены с помощью выпадающего меню, но большинство необходимо набирать на клавиатуре.

На первом этапе необходимо изучить три вещи**:** создание и изменение элементов, сохранение и удаление объектов из памяти, и доступ справке (помощь).

1.1. Оператор <-

R – *объектно-ориентированный язык*: переменные, данные, матрицы, функции, результаты, и т.д., хранятся в оперативной памяти компьютера в форме объектов, которые имеют имя. Для отображения его значения необходимо напечатать название объекта. Например, для отображения значения объекта n :

```
> n
[1] 10
```

Цифра 1 в скобках указывает, что отображается первый элемент n, который имеет значение 10.

Для того чтобы присвоить значение объекту используется символ *"<-"*. Этот символ пишется вместе со знаком минус так, чтобы они представляли стрелку, которая может быть направлена слева направо, или наоборот:

```
> n <- 15
> n
[1] 15
> 5 -> n
> n
[1] 5
```

Значение также, может быть результатом арифметического выражения:

```
> n <- 10+2
> n
[1] 12
```

Можно просто напечатать выражение, не присваивая ему название, тогда результат будет отображен на экране, но не сохранен в памяти:

```
> (10+2) *5
[1] 60
```

1.2. Отображение и удаление объектов в памяти

Функция **ls()** показывает объекты, находящиеся памяти, отображая только их названия.

```
> name <- "ВОЛГА"; n1 <- 10; n2 <- 100; m <- 0.5
```

```
> ls()
[1] "m" "n1" "n2" "name"
```

При записи нескольких команд в одной строке, для того чтобы отделить одну команду от другой используется символ "**;**" (точка с запятой).

Если объектов в памяти много, может быть целесообразно, перечислить только те объекты, в названии которых содержатся определенные символы. Это может быть сделано с помощью параметра pattern (который может быть сокращен: pat):

```
> ls(pat = "m.")
[1] "m" "name"
```

Если мы хотим ограничить список объектов, например, названиями которые начинаются с этого символа:

```
> ls(pat = " ^ m.")
[1] "m"
```

Чтобы показать характеристики объектов, можно использовать функцию **ls.str()*:***

```
> ls.str()
n1: num 10
n2: num 100
name: chr "name"
```

По умолчанию, **ls.str()** отображает все характеристики объекта, включая столбцы данных, матрицы и списки, которые могут привести к очень длинному выводу для составных объектов.

```
> M. <-data.frame (n1, n2, m)
> ls.str(pat = "M.")
M: ‘data.frame’: 1 Обь. из 3 переменных:
$ n1: цифровой 10
$ n2: цифровой 100
$ м.: цифровой 0.5
```

Для определения уровня подробностей отображения составных объектов с помощью функции ***ls.str()*** используется параметр ***max.level***

```
> ls.str(pat = "M.", max.level =-1)
M: ‘data.frame’: 1 Обь. из 3 переменных:
```

Для удаления объекта из памяти, используется функция **rm()*:*** **rm(x)** удалит объект **x, rm(x, y)** - объекты x и y , **rm(list=ls())**– удалит все объекты. Упомянутые выше варианты для функции **ls()** можно также использовать, выборочного удаления объектов:

```
>rm (list=ls(pat = "m"))
```

1.3. Вызов справки

Справка R дает некоторую очень полезную информации относительно того, как использовать функции.

Справку в формате html вызывают, печатая:

```
> help.start()
```

Поиск с ключевыми словами возможен со справкой html. Поиск функций может быть сделан при помощи оператора apropos ("что"), который выводит список имен функции с "что" в их имени:

```
> apropos("anova")
 [1] "anova" "anova.glm" "anova.glm.null"
"anova.glmlist"
 [5] "anova.lm" "anova.lm.null" "anova.mlm"
"anovalist.lm"
 [9] "print.anova" "print.anova.glm" "print.anova.lm"
"stat.anova"
```

Справка доступна и в текстовом формате для конкретной функции, например:

```
> ?lm
```

отображает файл справки для функции ***lm()***. Функция ***help(lm)*** или ***help("lm")*** имеет тот же самый эффект. Эта последняя функция используется, чтобы обратиться к справке с нетрадиционными символами:

```
>? *
Ошибка: синтаксическая ошибка
> help ("*")
Арифметика package:base R Документация
Арифметические Операторы.
```

2. Данные в R

2.1. Объекты

R работает с объектами, которые имеют два встроенных атрибута: ***тип данных*** и ***длина***. **Тип данных** – вид элемента; имеются четыре типа данных: *num(числовой)*, *char(символьный)*, *complex(комплексный)* и *logical(логический)*. **Длина** - общее количество элементов объекта. Приведем список возможных типов данных для различных объектов

Возможные типы данных для R объектов:

Объект	Возможные типы данных	Использование в объекте нескольких типов данных
vector	числовой, символьный, комплексный, логический	Нет
factor	числовой, символьный	Нет
array	числовой, символьный, комплексный, логический	Нет
matrix	числовой, символьный, комплексный, логический	Нет
data.frame	числовой, символьный, комплексный, логичный	Да

Ts	числовой, символьный, комплексный, логический	Да
List	числовой, символьный, комплексный, логический, функция, выражение, формула	Да

vector (вектор) – обычная переменная.

factor (фактор) – категорийная переменная.

array (массив) – таблица с *k* измерениями

matrix (матрица) – частный случай массива с $k = 2$. У массива и матрицы все элементы одного и того же типа.

Data.frame – таблица состоящая из нескольких векторов одинаковой длины, но возможно различных типов.

Ts – набор данных временного ряда, содержащий дополнительные атрибуты, такие как: частота и дата.

Среди несвойственных атрибутов объекта, может быть: *dim* (размерность) - который соответствует измерениям многомерного объекта. Например, матрица с 2 строками и 2 столбцами имеет для **dim**, пару значений [2,2], но его *длина* – 4.

R также различает в названий объектов, заглавные и строчные буквы. Так, *x* и *X* используются к различным объектам:

```
> x <-1; X <-10
> ls ()
[1] "X" "x"
> X
[1] 10
> x
[1] 1
```

2.2. Чтение данных из текстового файла

R может читать данные, сохраненные в текстовом (ASCII) файле. Для этого используются три функции: **read.table()** (которая имеет два варианта: **read.csv()** и **read.csv2()**), **scan()** и **read.fwf().**

Например, если мы имеем файл data.dat, то для того чтобы его прочитать можно набрать:

```
> mydata <-read.table ("data.dat")
```

mydata тогда будет объектом вида *data.frame*, и каждая переменная будет называться, по умолчанию, *V1, V2...* и к ним можно обратиться индивидуально *mydata$V1, mydata$V2..., или mydata["V1"], mydata["V2"]..., или mydata[,1], mydata[,2], …*

Однако, есть различия: *mydata$V1* и *mydata[,1]* – векторы, тогда как *mydata["V1"]* - data.frame.

У функции **read.table()** есть несколько параметров:

```
read.table(файл, header=FALSE, sep = "",
quote = "\"'", dec = ".", row.names =, col.names =,
as.is=FALSE, na.strings = "NA", skip=0,
fill = ! blank.lines.skip, check.names=TRUE,
strip.white=FALSE, blank.lines.skip = TRUE)
```

Параметры:

файл	название файла пишется (в кавычках " "), можно использоваться путь (символ \ не допустим, и должен быть заменен на /).
header	логический (ЛОЖЬ или ИСТИНА) указывает, содержит ли файл названия переменных в начале строки.
sep	разделитель полей используемый в файле, например sep = "\t", если это - табуляция.
quote	символы, используемые для обозначения переменных символьного типа.
dec	символ, используемый для отделения дробной части числа.
row.names	вектор с названиями строк, которые могут быть векторами символьного типа, или числового (или название) переменной файла (по умолчанию: 1, 2, 3...).
col.names	вектор с названиями колонок (по умолчанию: V1, V2, V3...).
as.is	указывает преобразовать символьных значения в факторы (если ЛОЖЬ) или сохраняет их как символы (ИСТИНА).
na.strings	показывает какое значение строки считать отсутствием данным (по умолчанию - NA). Пустое значение в числовой колонке тоже считается отсутствием данных.
skip	число строк, которые будут пропущены перед чтением данных
fill = TRUE	если в файле все поля заполнены.
check.names	если ИСТИНА, то проверяется, допустимость имен переменных для **R**
strip.white	(условное выражение к sep), если ИСТИНА, то удаляются дополнительные пробелы до и после символьных переменных
blank.lines .skip	Если имеет значение TRUE то пустые строки в файле игнорируются по умолчанию параметр имеет значение TRUE Мы можем изменить значение на FALSE, которое используется совместно с параметром fill = TRUE

Для ввода из файлов имеющих определенный формат используются команды **read.csv** и **read.csv2** которые имеют следующие значения по умолчанию:

read.csv (файл, header = ИСТИНА, sep =",""
quote = " \ "", dec = ". "...)

read.csv2 (файл, header = ИСТИНА, sep = ";"
quote = " \ "", dec = ", "...)

Функция **scan()** более гибкая, чем **read.table()** и имеет больше параметров. Главное отличие в том, что, при вводе можно указать тип переменных, например:

> mydata <-scan("data.dat", what=list (" ", 0,0))

в файле *data.dat* три переменные, первая имеет символьный тип и следующие две - числовой тип.

Формат **scan()** следующий:

> scan (file = "", what=double(0), nmax=-1, n=-1,
sep="", quote=if(sep == "\n") " " else " ' \ "",
dec = ". ", skip=0, nlines=0, na.strings = "NA",
flush=FALSE, strip.white=FALSE, quiet=FALSE)

Параметры:

file	название файла пишется (в кавычках " "), можно использоваться путь (символ \ не допустим, и должен быть заменен на /), если file = " ", то данные вводят с клавиатуры .
what	определяет типы данных
nmax	количество читаемых данных, или, если what описывает список, количество читаемых строк (по умолчанию, scan читает данные до конца файла)
n	количество читаемых данных, (по умолчанию все)
sep	разделитель полей используемый в файле
quote	символы, используемые для обозначения переменных символьного типа
dec	символ, используемый для отделения дробной части числа.
skip	число строк, которые будут пропущены перед чтением данных
nlines	количество читаемых строк
na.strings	присваивает значения, отсутствующим данным (показывает как NA)
flush	логический, если TRUE, то scan переходит на следующую строку, как только достигнет последнего столбца

strip.white	(условное выражение к sep), если TRUE, то удаляются дополнительные пробелы до и после символьных переменных
quiet	логический, если FALSE, scan отображает строку, показывающую, какие поля были прочитаны

Функция **read.fwf()** используется, для чтения данных имеющих определенный фиксированный формат:

read.fwf (файл, widths, sep = "\t ", as is=FALSE, skip=0, row.names, col.names)

Например, если файл *data.txt* имеет следующий вид:

A1 501 2
A1 551 3
B1 601 4
B1 651 5
C1 701 6
C1 751 7

То его можно прочитать следующим образом:

```
> mydata <-read.fwf("data.txt", widths=c(1,4,3))
> mydata
  V1 V2 V3
1 A 1.50 1.2
2 A 1.55 1.3
3 B 1.60 1.4
4 B 1.65 1.5
5 C 1.70 1.6
6 C 1.75 1.7
```

2.3. Сохранение текстовых данных

Для сохранения данных используются функции **write()**, **write.table()**, **write.matrix().**

Функция ***write(x, file="data.txt")*** сохраняет объект *x* в файле *data.txt*. У данной функции имеются еще два параметра:

nc (или *ncol*), - который определяет число столбцов в файле (по умолчанию *nc=1*, если *x* символьного типа, *nc=5* для других типов), и параметр

append – (логического типа), если он имеет значение *ИСТИНА* ,то данные добавляются в уже существующий файл, а если имеет значение *ЛОЖЬ*, (значение по умолчанию), то старые данные удаляются а новые заносятся в уже пустой файл.

Функция **write.table()** сохраняет данные в файле формата *data.frame*. Формат которой:

write.table(x,файл,append=FALSE, quote=TRUE, sep = " ", eol = "\n", na = "NA", dec = ".", row. names=TRUE, col. names=TRUE)

Параметры:

sep	разделитель полей используемый в файле
dec	символ, используемый для десятичной дроби
col.names	(логический) определяет, написаны ли названия столбцов в файле
row.names	(логический) определяет используется ли идентификатор для названий строк
quote	логический или числовой вектор; если TRUE то переменные символьного типа записываются с " "; если числовой вектор, его элементы дают индексы столбцов, которые записаны в " ". В обоих случаях, названия строк и столбцов также отображаются с " " если они написаны.
na	символ, который используется для пропущенных данных
eol	символ, который используется в конце каждой строки ("\n" - перевод каретки(курсора))

Пример

Для записи CSV-файла для ввода данных в Excel или Dbase вы должны использовать запись

```
write.table(x, file = "foo.csv", sep = ",", col.names = NA)
```

и для ввода этого файла обратно в R необходимо написать

```
read.table("file.csv", header = TRUE, sep = ",", row.names=1)
```

Функция **write.matrix()** сохраняет матрицу или объект типа **data.frame** в файл или выводит на экран, используя обозначение столбца и схему относительно размещения столбцов.

write.matrix (x, file = " ", sep = " ", blocksize)

Параметры:

x	матрицу или объект типа data.frame
file	название выходного файла. Значение по умолчанию (" ") - пульт.
sep	разделитель между столбцами.
blocksize	если параметр написан и имеет положительное значение, то вывод пишется в блоках из *blocksize* строк. Выберайте как можно большее значение совместимое с доступным объемом памяти

2.4. Таблицы произвольной структуры

Отображать таблицы большого размера в форме регулярного массива не всегда удобно. В категорическом анализе данных, такая информация часто представляется в форме из ограниченных двумерных массивов со строками и столбцами, определяющими комбинацию из уровней фактора, соответствующих числу ячейки. Эти строки и столбцы - "неровные" в смысле, что значения отображаются, когда они изменяются, с условием, что строки читаются сверху вниз и столбцы читаются слева направо.

В R, такие таблицы сопряженности признаков могут быть созданы, используя функцию ftable, которая создает объекты класса ″ftable″ с соответствующим методом печати. Как простой пример, рассмотрим R стандартный набор данных UCBAdmissions, которым является 3-мерная таблица сопряженности признаков, следующая из классификации претендентов.

```
> data (UCBAdmissions)
> ftable (UCBAdmissions)
```

	Dept	A	B	C	D	E	F
принимать	Пол						
допустимы	мужчина	512	353	120	138	53	22
	женщина	89	17	202	131	94	24
Отклонены	мужчина	313	207	205	279	138	351
	женщина	19	8	391	244	299	317

Напечатанное представление более удобно, чем отображение данных как 3-мерный массив.

Для того чтобы читать подобные таблицы используется функция **read.ftable**

read.ftable(file,sep = "", quote = "\"", row.var.names, col.vars, skip = 0)

Параметры:

file	символьная строка названия файла.’’’’’’ Указывает на ввод со стола оператора и туда же вывод для записи.
Sep	разделитель строк.Значения на каждой строке файла разделены этой строкой.
quote	символьная строка.
row.var.names	символьный вектор с названиями строк переменных, в случае, если они не могут быть определены автоматически.
col.vars	список, дающий названия и уровни столбцов переменных, в случае если они не могут быть определены автоматически.
skip	определяет число строк в файле, которые не используются при чтении данных.

Она имеет дополнительные параметры для различных вариантов: например, как информация значений строк и столбцов называется и как представлены уровни.

Пример:

```
file <- tempfile()
cat("              отношения \n",
    "Соц  различия    Yes  No\n",
    "Город Муж         43 134\n",
    "      Жен       26 149\n",
    "Село Муж          29  23\n",
    "      Жен       22  36\n",
    file = file)
file.show(file)
ft <- read.ftable(file)
ft
unlink(file)
```

Для записи подобных таблиц используется команда **write.ftable**:

write.ftable(x, file = "", quote = TRUE, digits = getOption("digits"))

Параметры:

x	объект класса `"ftable"'.
Digits	целое число,обозначающее число цифр для использования в x.

Эти таблицы могут быть преобразованы в стандартные таблицы сопряженности признаков в форме массива, используя функцию **as.table**.

as.table(x, ...)

оставшиеся параметры будут зависеть от выбранного метода.

2.5. Загрузка и запись данных

Для записи группы объектов в двоичной форме используется функция save():

save(..., list = character(0), file = "", ascii = FALSE, version = NULL, envir = parent.frame(), compress = FALSE)

Параметры:

...	названия объектов,которые будут сохранены.
list	символьный вектор,содержащий названия объектов,которые будут сохранены.
file	название файла,где будут сохранены данные.
ascii	если TRUE, то данные будут записываться в формате ASCII. Это необходимо для обмена данных между машинами различного типа. Значение по умолчанию является FALSE,

которое означает запись в двоичном файле.

version	версия формата рабочей области. NULL означает,что используется текущий формат. Версия, для R от 0.99.0 до R 1.3.1 равна 1. По умолчанию формат для R от 1.4.0 равен 2.
envir	путь для поиска сохраненных объектов.
compress	логический.Определяет должно ли использоваться сжатие к сохраненному файлу. Игнорируемый для версии 1 формата рабочей области.
safe	логический.Если TRUE, временный файл используется для создания сохраненной рабочей области

Функция **save.image ()** – сокращенный аналог команды save()с параметрами:

save.image(file = ".RData", version = NULL, ascii = FALSE,compress = FALSE, safe = TRUE)

Сохраненные данные (которые называются image), позже могут быть загружены в память с помощью команды **load()**

load(file, envir = parent.frame())

Параметры:

file	символьная строка - имя файла загрузки
envir	путь,где расположены данные

Пример

```
> save (x, y, z, file = "Mystuff. RData").
> save(list=ls (all=TRUE),file = ". RData").
```

Для облегчения передачи данных между машинами, может быть использован параметр *ascii=TRUE*. Таким образом, сохраненные данные (которые называются image), позже могут быть загружены в память с помощью команды

> load ("Mystuff. RData").

2.6. Генерация (создание) данных

2.6.1. Регулярные последовательности

Регулярная последовательность целых чисел, например от 1 до 30, может быть создана следующим оператором:

```
> x <-1:30
```

Результирующий вектор *x* имеет 30 элементов. Оператор ‘:‘ имеет приоритет над арифметическими операторами:

```
> 1:10-1
[1] 0 1 2 3 4 5 6 7 8 9
> 1: (10-1)
[1] 1 2 3 4 5 6 7 8 9
```

Функция **seq()** создает последовательности действительных чисел:

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

– где первый параметр указывает начальное число последовательности, второй конечное число, и третье – приращение. Можно использовать также следующий вариант:

```
> seq (length=9, from=1, to=5)
 [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

Также возможно напечатать непосредственно значения, используя функцию **c():**

```
> c (1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5)
 [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

которая дает такой же результат.

Функция **rep()** создает вектор с одинаковыми элементами:

```
> rep (1, 30)
 [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Функция sequence() создает ряд последовательностей целых чисел каждая, из которых заканчивается числом, которое является параметром функции:

```
> sequence (4:5)
 [1] 1 2 3 4 1 2 3 4 5
> sequence (c (10,5))
 [1] 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5
```

Функция **gl(k,n)** создает правильный ряд факторных переменных, где *k* – количество уровней (или классов) и *n* – количество чисел в каждом уровне. Дополнительно могут использоваться два параметра: *length* – длина, чтобы определить количество чисел в общей последовательности и *labels*, чтобы определить названия коэффициентов (факторов). Например:

```
> gl (3,5)
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
 Уровни: 1 2 3
> gl (3,5,30)
 [1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
 Уровни: 1 2 3
> gl (2,8, label=c ("Управление", "Обрабатывает"))
 [1] Управление Управление Управление Управление Управление Управление Управление Управление Обрабатывает
 [10] Обрабатывает, Обрабатывает, Обрабатывает, Обрабатывает, Обрабатывает, Обрабатывает, Обрабатывает
 Уровни: Управление Обрабатывает
> gl (2, 1, 20)
 [1] 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2 1 2
 Уровни: 1 2
```

```
> gl (2, 2, 20)
[1] 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2 1 1 2 2
Уровни: 1 2
```

Функция **expand.grid()** создает объект типа **data.frame** со всевозможными комбинациями векторов или коэффициентов(факторов) данных:

```
> expand.grid (h=seq (60,80,10), w=seq (100 300 100),
sex=c ("Входящий в другую деталь", "Охватывающий"))
    h   w  sex
 1 60 100 Входящий в другую деталь
 2 70 100 Входящий в другую деталь
 3 80 100 Входящий в другую деталь
 4 60 200 Входящий в другую деталь
 5 70 200 Входящий в другую деталь
 6 80 200 Входящий в другую деталь
 7 60 300 Входящий в другую деталь
 8 70 300 Входящий в другую деталь
 9 80 300 Входящий в другую деталь
10 60 100 Охватывающий
11 70 100 Охватывающий
12 80 100 Охватывающий
13 60 200 Охватывающий
14 70 200 Охватывающий
15 80 200 Охватывающий
16 60 300 Охватывающий
17 70 300 Охватывающий
18 80 300 Охватывающий
```

2.6.2. Случайные последовательности

R позволяет генерировать набор случайных данных для большого количества функций плотности вероятности. Эти функции имеют следующий вид:

```
> rfunc (n, p [1], p [2]...)
```

где *func* определяет закон вероятности, *n* - число данных и *p[1], p[2]*- значения параметров закона. Следующая таблица отображает значения параметров для каждого закона, и возможные значения по умолчанию (если ни одно значение по умолчанию не обозначено, это означает, что параметр должен быть определен пользователем).

Закон	Параметры
Гауссовское распределение	*rnorm (n, mean=0, sd=1)*
Экспоненциальное распределение	*rexp (n, rate=1)*
Гамма распределение	*rgamma (n, shape, scale=1)*
Распределение Пуассона	*rpois (n, lambda)*

Распределение Вейбула	*rweibull (n, shape, scale=1)*
Распределение Коши	*rcauchy (n, location=0, scale=1)*
Бета распределение	*rbeta (n, shape1, shape2)*
Распределение Стьюдента(t)	*rt(n, df)*
Распределение Фишера(F)	*rf (n, df1, df2)*
Распределение Пирсона (X^2)	*rchisq (n, df)*
Биноминальное распределение	*rbinom (n, size, prob)*
Геометрическое распределение	*rgeom (n, prob)*
Гипергеометрическое распределение	*rhyper (nn, m, n, k)*
Логистическое распределение	*rlogis (n, location=0, scale=1)*
Логнормальное распределение	*rlnorm (n, meanlog=0, sdlog=1)*
Отрицательное биноминальное распределение	*rnbinom (n, размер, prob)*
Униформальное распределение	*runif (n, min=0, max=1)*
Распределение Вайлкоксона	*rwilcox (nn, м., n), rsignrank (nn, n)*

Обратите внимание, что все эти функции можно использовать, заменяя символ *r* на *d*, *p* или *q*, тогда вычисляем плотность вероятности **dfunc (x)**,кумулятивная плотность вероятности **pfunc (x**), и значения квантилей **qfunc (p), 0 <p <1**).

2.7. Управление объектами

2.7.1. Доступ к определенному значению объекта

Чтобы обратиться, например, к третьему значению вектора **x**, необходимо напечатать **x[3]**.

Если x - матрица или *data.frame*, то значение i-ой строки и j-ого столбца может быть получено: *x[i,j]*. Для замены всех значений третьего столбца, можно напечатать:

```
> x[, 3] <-10.2
```

Эта система индексации применяется к массивам, так что количество индексов совпадает с размерностью массива (например, трехмерный массив: **x[i,j,k], x[,,3]**). Необходимо отметить, что индексация производится с прямыми скобками [], тогда как круглые скобки используются для параметров функции :

```
>x(1)
Error: couldn't find function "x"(Ошибка: функция "x"
не найдена)
```

Индексация может использоваться, чтобы исключить одну или несколько строк или столбцов. Например, **x[-1],** будет пропускать первую строку, или **x[-c (1,15)**,] сделает то же самое для 1-ой и 15-ой строк.

Для векторов, матриц и массивов, можно, обратиться к значениям элемента используя для индекса выражения сравнения:

```
> x <-1:10
> x
[1] 1 2 3 4 5 6 7 8 9 10
> x [x> = 5] <-20
> x
[1] 1 2 3 4 20 20 20 20 20 20
> x [x == 1] <-25
> x
[1] 25 2 3 4 20 20 20 20 20 20
```

В **R** используется шесть операторов сравнения: **<(меньше),> (больше), <= (меньше или равно),> = (больше или равно), == (равняются) и ! (отличный от).**

Операторы возвращают переменную логического типа (ИСТИНА или ЛОЖЬ).

2.7.2. Арифметика и простые функции

В **R** существуют много функций для управления данными. Самая простая из них **c()**:

```
> c (1:5, seq (10, 11, 0.2))
[1] 1.0 2.0 3.0 4.0 5.0 10.0 10.2 10.4 10.6 10.8 11.0
```

Вектора могут управляться классическими арифметическими выражениями:

```
> x <-c (1,2,3,4)
> y <-c (1,1,1,1)
> z <-x + y
> z
[1] 2.0 3.0 4.0 5.0
```

С векторами различной длины также можно выполнять различные арифметические действия; в этом случае, самый короткий вектор будет использоваться в вычислениях несколько раз при этом длина длинного вектора должна быть кратна длине короткого вектора.

Например:

```
> x <-c (1,2,3,4)
> y <-c (1,2)
> z <-x+y
> z
[1] 2 4 4 6
```

```
> x <-c (1,2,3)
> y <-c (1,2)
> z <-x+y
Warning message:
longer object length
is not a multiple of shorter object length in: x + y
```
(Предупреждающее сообщение:Объект большей длины не кратен объекту меньшей длины в: x + y)
```
> z
[1] 2 4 4
```

В этом случае **R** выдает предупреждающее сообщение, а не ошибку. Таким образом, операция может быть выполнена.

Если мы хотим добавить (или умножить) одно и то же значение ко всем элементам вектора, то это можно сделать следующим образом:

```
> x <-c (1,2,3,4)
> a<-10
> z <-a*x
> z
[1] 10 20 30 40
```

Для создания выражений могут быть использованы следующие арифметические операторы: **+,-, *,/, и ^** *(для степеней),* **%%** (x %% y - остаток от деления), и **%/%** (x %/% y для деления нацело (возвращает целую часть от деления)).

В выражении можно использовать также функции.

Некоторые из функций представлены в следующей таблице:

sum(x)	Сумма элементов объекта x
prod(x)	Произведение элементов объекта x
max(x)	Максимальное значение из объекта x
min(x)	Минимальное значение из объекта x
which.max(x)	Индекс максимального значения объекта x
which.min(x)	Индекс минимального значения объекта x
range(x)	Минимальное и максимальное значения объекта x
length(x)	Число элементов в объекте x
mean(x)	Среднее значение элементов объекта x
median(x)	Медиана объекта x
var(x) или cov(x)	Дисперсия элементов(рассчитанная на n-1); если x матрица или data.frame, то вычисляется дисперсионно-ковариационная матрица.
cor(x)	Корреляция матрицы x.
var(x,y) или cov(x)	Ковариация между x и y или между столбцами x и столбцами y, если x и y матрицы или data.frames
cor(x,y)	Линейная корреляция между x и y или корреляционную матрицу, если они матрицы или data.frames

Все эти функции возвращают единственное значение (вектор длины 1),

кроме **range()** и **var()**, **cov()** и **cor(),** которые могут возвращать матрицу.

Следующие функции возвращают более сложные результаты:

round(x,n)	округляет элементы x до n знаков после запятой
rev(x)	перестановка элементов x в обратном порядке
sort(x)	сортирует элементы x в возрастающем порядке; сортировать в убывающем порядке rev(sort(x))
rank(x)	выдает классы элементов x
log(x,base)	вычисляет логарифм x с основным base
pmin(x,y,...)	вектор, в котором i-й элемент минимальный из x [i], y [i]...
pmax (x, y...)	вектор, в котором I-й элемент максимальный из x [i], y [i]...
cumsum(x)	вектор, i-й элемент которого является суммой от x [1] до x [i]
cumprod (x)	вектор, i-й элемент которого является произведением от x [1] до x [i]
cummin (x)	вектор, i-й элемент которого является минимальным из элементов от x [1] до x [i]
cummax (x)	вектор, i-й элемент которого является максимальный из элементов от x [1] до x [i]
match(x,y)	возвращает вектор той же длины что и вектор x с элементами x, которые находятся в y (иначе NA)
choose(n,k)	вычисляет комбинации *k* событий среди *n* повторений $\frac{n!}{k!\ n-k\ !}$
na.omit(x)	игнорирует наблюдения с отсутствующими данными (NA) (игнорирует соответствующую строку, если x является матрицей или data.frame)
na.fail(x)	выдает сообщение об ошибках, если x содержит NA (s)
table(x)	возвращает таблицу с номерами различных значений x (обычно для целых чисел или коэффициентов(факторов))
subset(x,...)	возвращает выборку x относительно критерия (...) в зависимости от режима x (обычно для сравнения: x$V1 <10); если x - data.frame, опция select позволяет идентифицировать переменные, которые будут сохранены

2.7.3. Операции с матрицами

R имеет средства для вычисления матриц и управления ими. Матрица может быть создана при помощи функции **matrix()**:

```
> matrix (data=5, nr=2, nc=2)
             [, 1]  [, 2]
       [1,]  5      5
       [2,]  5      5
> matrix (1:6, nr=2, nc=3)
             [, 1]  [, 2]  [, 3]
       [1,]  1      3      5
       [2,]  2      4      6
```

Функции **rbind()** и **cbind()** связывают (объединяют) матрицы относительно строк или столбцов, соответственно:

```
> m1 <-matrix (data=1, nr=2, nc=2)
> m2 <-matrix (data=2, nr=2, nc=2)
> rbind (m1, m2)
             [,1]  [,2]
       [1,]  1     1
       [2,]  1     1
       [3,]  2     2
       [4,]  2     2
> cbind (m1, m2)
             [,1]  [,2]  [,3]  [,4]
       [1,]  1     1     2     2
       [2,]  1     1     2     2
```

Оператор произведения двух матриц – **"% * %"**. Например, рассматривая две матрицы m1 и m2 созданные выше:

```
> rbind (m1, m2) % * % cbind (m1, m2)
```

мы получим следующий результат:

```
             [,1]  [,2]  [,3]  [,4]
       [1,]  2     2     4     4
       [2,]  2     2     4     4
       [3,]  4     4     8     8
       [4,]  4     4     8     8
> cbind (m1, m2) % * % rbind (m1, m2)
             [,1]  [,2]
       [1,]  10    10
       [2,]  10    10
```

Транспонирование матрицы производится функцией **t()**; эта функция также подходит для *data.frame*.

Функция **diag()** может использоваться, чтобы извлечь или изменить диагональ матрицы, или для построения диагональной матрица.

```
> diag (m1)
[1] 1 1
> diag (rbind (m1, m2) % * % cbind (m1, m2))
[1] 2 2 8 8
> diag (m1) <-10
```

```
> m1
     [,1] [,2]
[1,]  10    1
[2,]  1     10
> diag (3)
     [,1] [,2] [,3]
[1,]  1     0     0
[2,]  0     1     0
[3,]  0     0     1
> v <-c (10,20,30)
> diag (v)
     [,1] [,2] [,3]
[1,]  10    0     0
[2,]  0     20    0
[3,]  0     0     30
> diag (2.1, nr=3, nc=5)
     [,1] [,2] [,3] [,4] [,5]
[1,] 2.1  0.0  0.0  0    0
[2,] 0.0  2.1  0.0  0    0
[3,] 0.0  0.0  2.1  0    0
```

3. Графики в R

R предлагает большое разнообразие графиков. Чтобы понять идею, можно напечатать demo (graphics). Просто невозможно описать все возможности **R** в отображении графиков, особенно когда каждый графическая функция имеет большое количество параметров.

Сначала несколько подробностей относительно того, как управлять графическими окнами.

3.1. Управление графическими окнами

3.1.1. Открытие новых графических окон

В том случае, когда вы задаете графическую функцию, открывается, графическое окно с требуемым графиком. Также можно *открыть другое окно с помощью функции* **x11()**:

```
> x11()
```

Окно открытое таким образом, становится активным окном, и последующие графики будут отображены в нем. *Чтобы узнать, какие графические окна открыты* в настоящее время, необходимо использовать функцию: **dev.list()**

```
>dev.list()
>windows windows
> 2 3
```

Числа, отображенные «**windows**» это номера окон, которые могут быть использованы, чтобы изменить активное окно:

```
> dev.set (2)
> windows
2
```

3.1.2. Разделение графического окна

Функция **split.screen()** делит активное графическое окно. Например, **split.screen (c(1,2))** делят окно на две части, которые могут быть вызваны **screen (1)** или **screen (2)**;

Функция **erase.screen()** стирает последний нарисованный график.

Функция **layout()** позволяет делать более сложное разделение: она делит активное графическое окно на несколько частей, где графики будут отображены последовательно. Например, деление окна на четыре равные части:

```
> layout (matrix (c (1,2,3,4), 2, 2))
```

где вектор дает числа подокон, и два числа 2 указывают на то, что это окно будет разделено на две строки и два столбца. Команда:

```
> layout (matrix (c (1,2,3,4,5,6), 3, 2))
```

создаст шесть подокон, в три строки и два столбца, тогда как:

```
> layout (matrix (c (1,2,3,4,5,6), 2, 3))
```

также создаст шесть подокон, но в две строки и три столбца.

Подокна могут иметь различные размеры:

```
> layout (matrix (c (1,2,3,3), 2, 2))
```

откроет два подокна в строчку в левой половине окна, и три подокна в правой половине.

```
> layout (matrix (c (1,1,2,1), 2, 2), c (3,1), c (1,3))
```

вектора с (3,1) и с (1,3) дают представления относительно размеров подокон.

Чтобы визуализировать разделение, созданное *layout ()* перед построением графика, можно использовать функцию **layout.show(2)**, если, например, два подокна были определены.

3.2. Графические функции

plot(x)	График значений x (на оси y), упорядоченные на оси x
plot(x,y)	Двумерный график
sunflowerplot(x,y)	Двумерный график(аналог plot()), но точки на графике закрашены.
piechart(x)	Круговая диаграмма
boxplot(x)	Ящик с усами
coplot (x~y|z)	Двумерный график x и y для каждого значения z(если z-коэффициент(фактор))

interaction.plot (f1, f2, x)	Если f1 и f2 - коэффициенты(факторы), составляет график средних значений y (на *оси Y*) относительно значений f1 (на *оси X*) и f2 (различные кривые); параметр fun = позволяет выбирать основную статистику y (по умолчанию fun=mean)
matplot (x, y)	Двумерный график первого столбца x *с* первым из y, второй из x со вторым из y, и т.д.
dotplot (x)	Если x - data.frame, производится построение Кливлендского точечного графика.
pairs(x)	Если x - матрица или data.frame, рисует все возможные двумерные графики между столбцами x
plot.ts (x)	Если x - объект класса ts,то отображает график x относительно времени, x может быть многомерен, но ряды должны иметь ту же самую частоту и дату
ts.plot (x)	Аналог предыдущему, но если x многомерен, ряды могут иметь различные даты ,а частоты те же самые
hist(x)	Гистограмма частот x
barplot (x)	Гистограмма значений x
qqnorm (x)	Множество значений x относительно значений, ожидаемых согласно нормальному закону распределения
qqplot (x, y)	Множество значений y относительно множества x
contour(x,y,z)	Создает график контура (используются интерполированные данные), x и y должны быть векторами и z должна быть матрица так, чтобы dim (z) =c (длина (x), длина (y))
image(x,y,z)	Аналог предыдущего, но в цвете
persp(x,y,z)	Трехмерный график

Для каждой функции, параметры могут быть найдены при помощи сетевой справки **R**. Некоторые из этих параметров являются идентичными для нескольких графических функций; вот - основные (с их возможным значением по умолчанию):

Add = FALSE если ИСТИНА новый график отображается вместе с предыдущим (если он существует)

Axes = TRUE если ЛОЖЬ оси не отображаются

Type = "p" определяет тип графика, "*p*":точки, "*l*": линии, "*b*":точки, соединенные линиями, "*o*": идентификатор, но линии над точками, "*h*": вертикальные линии, "*s*": шаги, данные представлены вершиной вертикальных строк, "*S*": идентификатор, но данные представлены основанием вертикальных строк.

Xlab =, примечания для осей, должны быть переменными

ylab=	символьного типа (любая символьная переменная, или строка в пределах " ")
Main=	основной заголовок, должен быть переменной символьного типа
sub =	подзаголовок (написанный в меньшем шрифте)

3.3. Команды управления графиками нижнего уровня

R имеет набор графических функций, которые применяются для уже существующий графиков: их называют команды управления графиками нижнего уровня. Укажем некоторые из них:

Points(x,y)	Добавляет точки
Lines(x,y)	Добавляет линии
Text(x,y,labels,…)	Добавляет текст в *labels* к координатам *(x,y)*; обычно используется*: plot(x,y,type="n")*; *text(x,y,names)*
Segments(x0,y0,x1, y1)	Рисует линию от точки(x0,y0) к точке(x1,y1)
Arrows(x0,y0,x1,y1 , angle=30, code=2)	Аналог предыдущей с курсором в точке(x0,y0), если *code=2*, в точке(x1,y1), если *code=1* или в обоих, если *code=3*; *angle*–угол под котором будет рисоваться стрелка (указывает направление)
Abline (a, b)	Рисует линию наклона *b* и прерывает в *a*
Abline (h=y)	Рисует горизонтальную линию
Abline (v=x)	Рисует вертикальную линию
Abline (lm.obj)	Рисует линию регрессии (из объекта *lm.obj*)
Rect (x1,y1,x2,y2)	Рисует прямоугольник границы которого-x1,x2,y1,y2
Poligon(x,y)	Рисует полигон, ограниченный точками с координатами x и y
Legend(x,y,legend)	Добавляет легенду к точке символом, указанным в легенде
Title()	Добавляет заголовок и произвольно подзаголовок
Axis(side,vect)	Добавляет ось в основании (*side=1*), слева (2), наверху (3), или справа (4); *vect* (дополнительный) дает абсциссу (или ординату), куда рисуются метки
Rug(x)	Рисует данные x на *оси X* как маленькие вертикальные линии
Locator(n,type="n" ,…)	Возвращает координаты (*x*, *y*) после того, как пользователь нажал на графике n -время мышью; также рисует символы (*type = "p"*) или линии (*type = "l"*); по умолчанию ничего не рисует (*type = "n"*)

Обратите внимание на возможность добавить изображение математического выражения на графике: **text (x, y, expression (...)),** где функция *expression()* преобразовывает аргумент в математическое уравнение согласно кодированию, используемому в наборе TeX.

Например:

<-text(x,y,expression (Uk [37] == over (1, 1+e ^ {-epsilon * (T-theta)})))) отобразит, на графике уравнение в точке с координатами (*x*, *y*).

Чтобы включать в выражение переменную, можно использовать функцию

substitute () вместе с функцией **as.expression ()**;

Например, включать значение *R2* (предварительно вычисленное и сохраненное в объекте по имени *Rsquared*):

```
<-text (x, y, as.expression (substitute (R^2 == r,
list (r=Rsquared))))
```

отобразит на графике в точке с координатами (x, y):

R^2 = 0.9856298

Чтобы отображать только три числа после запятой, можно изменить код следующим образом:

```
<-text(x,y,as.expression(substitute(R^2== r,
list(r=round(Rsquared,3)))))
```

который приведет к

R^2= 0.986

Наконец, чтобы записать R при помощи курсива:

```
<-text(x,y,as.expression(substitute(italic(R)^2==r,
      list(r=round(Rsquared,3)))))
```

R2 = 0.986

3.4. Графические параметры

В дополнение к командам управления графиками нижнего уровня, представление графика может быть улучшено с помощью графических параметров. Они могут использоваться даже как опции графических функций (но это работает не для всех). Функция par() изменяет постоянные графические параметры, то есть последующие графики будут строиться относительно параметров, указанных пользователем.

Например:

```
> par (bg = "yellow")
```

будет рисовать все последующие графики с желтым фоном. Есть 68 графических параметров, исчерпывающий список графических параметров может получить печатая

? par

Вот некоторые из них:

adj	управляет выравниванием текста (0 выровненный по левому краю, 0.5 центрированный, 1 выровненный по правому краю)
bg	определяет цвет фона
bty	управляет типом поля вокруг графика, допустимые значения: *"o", "l", "7", "c", "u"* или *"]"*(поле напоминает соответствующий символ); если *bty = "n"* поле не рисуется
cex	значение регулирует размер текстов и символов относительно значения по умолчанию
col	управляет цветом символов; что касается сех есть: *col.axis, col.lab, col.title, col.sub*
font	целое число, которое управляет стилем текста (*0: обычный, 1: курсив, 2: полужирный, 3: полужирные курсивы*); что касается сех есть: *font.axis, font.lab, font.title, font.sub*
las	целое число, которое управляет ориентацией примечаний на осях (*0: параллельно осям, 1:горизонтально, 2: перпендикулярно осям, 3: вертикально*)
lty	управляет типом линий, может быть целое число (*1: сплошная, 2: штриховая, 3: пунктирная, 4: штрих пунктирная, 5: longdash, 6:двойное подчеркивание*), или строка до восьми символов (между "0" и "9"), которая альтернативно определяет длину в точках или пикселях, отображая элементы и пробелы
lwd	числовой, который управляет шириной линий
mar	вектор из четырех числовых значений, которые управляют пространством между осями и границей рисунка. Формат *c (bottom,left,top,right)*, значения по умолчанию - c(5.1,4.1,4.1,2.1)
mfcol	вектор формы *c (nr, nc)*, который делит графическое окно как матрицу из *nr* строк и *nc* столбцов, графики тогда рисуются в столбцах
mfrow	То же самое, но графики рисуются в строках
pch	управляет типом символа, или целое число между 1 и 25, или любой единственный символ в пределах " "
ps	целое число, которое управляет размером в точках текстов и символов
pty	символ, который определяет тип области составления графика, *"s"*: квадрат, *"m"*: максимальное
xaxt	если *xaxt = "n"* *ось X* установлена, но не нарисована
yaxt	если *yaxt = "n"* *ось Y* установлена, но не нарисована

Список литературы.

1. **Гмурман В.Е.** Теория вероятностей и математическая статистика/ В.Е.Гмурман.М.:Высшая школа, 2000.-479с.
2. **Лакин Г.Ф.** Биометрия/ Г.Ф. Лакин. М: Высшая школа, 1990.-352с.
3. Теория вероятностей и математическая статистика/ Под редакцией В.А. Колемаева. М: Высшая школа, 1991.-400с.
4. **Гайдышев И.** Анализ и обработка данных: специальный справочник - СПб: Питер, 2001.-752с.
5. **Бейли Н**. Статистические методы в биологии/Н.Бейли.М.:Мир,1963.-272с.
6. **Гланц С.** Медико-биологическая статистика/ С. Гланц. М: Практика, 1999.-449с.
7. **E. Paradis.** R for begginers/ E. Paradis -2000