ПЕРМСКИЙ
ГОСУДАРСТВЕННЫЙ
НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ

**В. А. Шишкин**

# МАТЕМАТИЧЕСКИЕ ПАКЕТЫ: R

Perm University Press

Пермь 2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное
образовательное учреждение высшего образования
«ПЕРМСКИЙ ГОСУДАРСТВЕННЫЙ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ»

В. А. Шишкин

# МАТЕМАТИЧЕСКИЕ ПАКЕТЫ: R

*Допущено методическим советом
Пермского государственного национального
исследовательского университета в качестве
учебного пособия для студентов, обучающихся
по направлениям подготовки бакалавров
«Прикладная математика и информатика»,
«Бизнес-информатика»*


Пермь 2023

УДК 51: 004(075.8)
ББК 22.18я73
Ш655

**Шишкин В. А.**

Ш655 Математические пакеты: R [Электронный ресурс] : учебное пособие / В. А. Шишкин ; Пермский государственный национальный исследовательский университет. – Электронные данные. – Пермь, 2023. – 15,0 Мб ; 225с. – Режим доступа: http://www.psu.ru/files/docs/science/books/uchebnie-posobiya/SHishkin-Matematicheskie-pakety-R.pdf. – Заглавие с экрана.

ISBN 978-5-7944-4072-0

Статистический пакет R, рассматриваемый в курсе «Математические пакеты», предназначен для проведения статистической обработки данных и графического отображения полученных результатов. Пакет R можно использовать во всех учебных курсах, где есть необходимость в статистических вычислениях, работе с данными и моделировании, например, Data Mining (исследование закономерностей), Methods and Models for Multivariate Data Analysis, Methods of statistical research in Economics, имитационное моделирование, математическая статистика, модели и методы принятия решений, учёт неопределённости в моделях экономического поведения.

**УДК 51: 004(075.8)**
**ББК 22.18я73**

*Издается по решению ученого совета механико-математического факультета Пермского государственного национального исследовательского университета*

*Рецензенты:* кафедра информационных технологий в бизнесе НИУ ВШЭ – Пермь (профессор кафедры – канд. физ.-мат. наук, профессор ***А. П. Иванов***);

доцент кафедры прикладной информатики, информационных систем и технологий Пермского государственного гуманитарно-педагогического университета, канд. пед. наук, доцент ***И. В. Ильин***

ISBN 978-5-7944-4072-0

# Оглавление

# Используемые обозначения

❑ — признак завершения примера.

T — логическая константа «ИСТИНА».

F — логическая константа «ЛОЖЬ».

$\neg$ — логическое отрицание:

| $x$ | $\neg x$ |
|---|---|
| F | T |
| T | F |

$\wedge$ — логическая связка «И» (конъюнкция).

$\vee$ — логическая связка «ИЛИ» (дизъюнкция).

$\to$ в логических выражениях — импликация. В описании оператора $f\colon X \to Y$ связывает область определения ($X$) с областью значений ($Y$).

| $x$ | $y$ | $x \wedge y$ | $x \vee y$ | $x \to y$ |
|---|---|---|---|---|
| F | F | F | F | T |
| F | T | F | T | T |
| T | F | F | T | F |
| T | T | T | T | T |

$a \vdash b$ — $b$ является логическим следствием $a$ (если $a$ истинно, то $b$ тоже истинно).

$A \cup B$ — объединение множеств $A$ и $B$:

$$A \cup B = \{x \mid x \in A \wedge x \in B\}$$

$A \cap B$ — пересечение множеств $A$ и $B$:

$$A \cup B = \{x \mid x \in A \vee x \in B\}$$

$\mathbb{N}$ — множество натуральных чисел: $\mathbb{N} = \{1, 2, \dots\}$.

Расширенное множество натуральных чисел: $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$.

$\mathbb{Z}$ — множество целых чисел: $\mathbb{Z} = \{-\mathbb{N}\} \cup \mathbb{N}_0$.

$\mathbb{R}$ — множество вещественных чисел.

$\mathbb{C}$ — множество комплексных чисел.

$\operatorname{mes} X$ — мощность множеста $X$ (число элементов, если $X$ конечно).

$I$ — тождественный оператор, единичная матрица: $Ix = x$.

$\det A$ — определитель матрицы $A$.

$A^\intercal$ — транспонированная матрица $A$.

$A^{-1}$ — обратная к $A$ матрица: $AA^{-1} = A^{-1}A = I$.

$\xi \sim \mathcal{U}(a,b)$ — случайная величина $\xi$ имеет равномерное распределение в интервале $[0,1]$.

$\xi \sim \mathcal{N}(\mu,\sigma^2)$ — случайная величина $\xi$ имеет нормальное распределение с математических ожиданием $\mu$ и дисперсией $\sigma^2$.

$\xi \sim \mathcal{L}og\mathcal{N}(\mu,\sigma^2)$ — случайная величина $\xi$ имеет логарифмически нормальное (логнормальное) распределение с параметрами $\mu$ и $\sigma^2$.

$\xi \sim \mathcal{E}xp(\lambda)$ — случайная величина $\xi$ имеет экспоненциальное распределение с параметром $\lambda$.

$\xi \sim t(n)$ — случайная величина $\xi$ имеет $t$-распределение Стьюдента с $n$ степенями свободы.

$\xi \sim \chi^2(n)$ — случайная величина $\xi$ имеет распределение хи-квадрат с $n$ степенями свободы.

$\xi \sim \mathcal{F}(n_1,n_2)$ — случайная величина $\xi$ имеет $\mathcal{F}$-распределение Фишера с $n_1$ и $n_2$ степенями свободы.

$\xi \sim \text{i.i.d.}(\mu,\sigma^2)$ — независимые случайные величины $\xi_i$ имеют одинаковое распределение с математическим ожиданием $\mu$ и дисперсией $\sigma^2$.

$\mathsf{P}\{A\}$ — вероятность события $A$.

$\mathsf{E}\,\xi$ — математическое ожидание $\xi$.

$\mathsf{Var}\,\xi$ — дисперсия $\xi$.

$\sigma_\xi$ — среднеквадратическое (или стандартное) отклонение $x$.

$s_x^2$ — выборочная дисперсия $x$.

$s_x$ — выборочное среднеквадратическое отклонение $x$.

$\mathrm{Cov}(\xi,\eta)$ — ковариация между $\xi$ и $\eta$.

$\mathsf{L}$ — оператор запаздывания: $\mathsf{L}\,y_i = y_{i-1}$.

$\Delta$ — разностный оператор: $\Delta y_i = y_i - y_{i-1}$.

$\delta_{ij}$ — символ Кронекера:

$$\delta_{ij} = \begin{cases} 1, & i = j, \\ 0, & i \neq j. \end{cases}$$

# Глава 1

# Введение

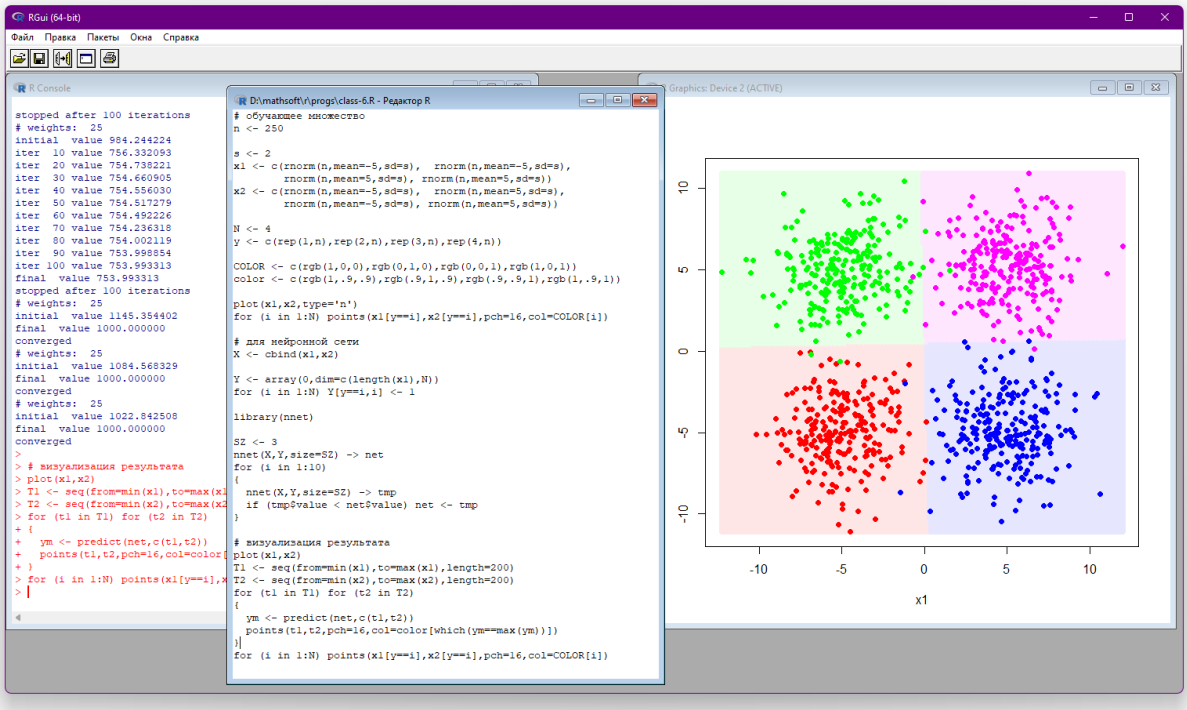



Рис. 1.1: Графическая оболочка R (Rgui) в среде Windows 11

R — свободное программное обеспечение, предназначенное для выполнения статистических расчётов и отображения результатов в виде графиков. Язык R был разработан в середине 90-х годов XX века как свободный аналог языка программирования S.

R позволяет:

- работать интерактивно, выполняя команды в консоли в режиме командной строки;
- выполнять наборы команд (скрипты);
- создавать программы, используя императивный, объектно-ориентированный и функциональный подходы.

## 1.1 Установка R

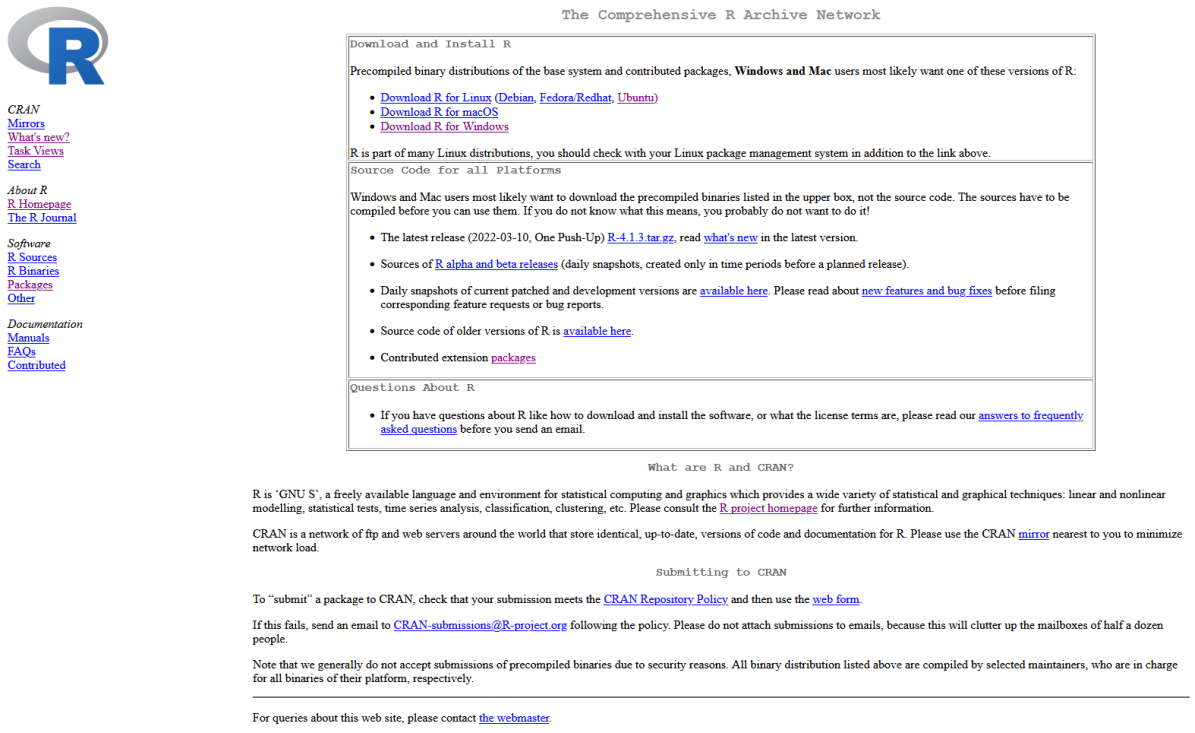
Для загрузки дистрибутива R следует на сайте www.r-project.org зайти в раздел CRAN — The Comprehensive R Archive Network, полный архив R (рис. 1.2) и выбрать ближайшее зеркало,


Рис. 1.2: Страница www.r-project.org

например, mirror.truenetwork.ru/CRAN (рис. 1.3). В зависимости от того, какая операционная система установлена на рабочем компьютере, выбирается подходящий вариант загрузки, например, Download R for Windows (рис. 1.4). Отсюда можно перейти в подкаталог base, содержащий ссылку на дистрибутив R (рис. 1.5), или в подкаталог contrib, из которого можно загрузить бинарные сборки пакетов расширения (рис. 1.6).

## 1.2 Пакеты расширения

Множество исследователей использует R для решения разнообразных задач. Если имеется некоторая проблема, связанная с анализом данных, то есть большая вероятность, что эта проблема уже возникала перед другими людьми, была решена, и её решение было оформлено



CRAN
Mirrors
What's new?
Task Views
Search

About R
R Homepage
The R Journal

Software
R Sources
R Binaries
Packages
Other

Documentation
Manuals
FAQs
Contributed

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux (Debian, Fedora/Redhat, Ubuntu)
- Download R for macOS
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2022-03-10, One Push-Up) R-4.1.3.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests or bug reports.
- Source code of older versions of R is available here.
- Contributed extension packages

Questions About R

- If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

What are R and CRAN?

R is 'GNU S', a freely available language and environment for statistical computing and graphics which provides a wide variety of statistical and graphical techniques: linear and nonlinear modelling, statistical tests, time series analysis, classification, clustering, etc. Please consult the R project homepage for further information.

CRAN is a network of ftp and web servers around the world that store identical, up-to-date, versions of code and documentation for R. Please use the CRAN mirror nearest to you to minimize network load.

Submitting to CRAN

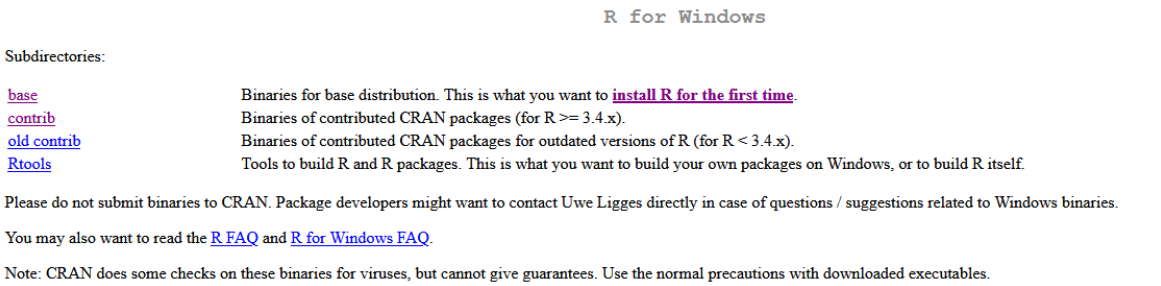
To "submit" a package to CRAN, check that your submission meets the CRAN Repository Policy and then use the web form.

If this fails, send an email to CRAN-submissions@R-project.org following the policy. Please do not attach submissions to emails, because this will clutter up the mailboxes of half a dozen people.

Note that we generally do not accept submissions of precompiled binaries due to security reasons. All binary distribution listed above are compiled by selected maintainers, who are in charge for all binaries of their platform, respectively.

For queries about this web site, please contact the webmaster.

Рис. 1.3: Зеркало CRAN: `mirror.truenetwork.ru/CRAN`



R for Windows

Subdirectories:

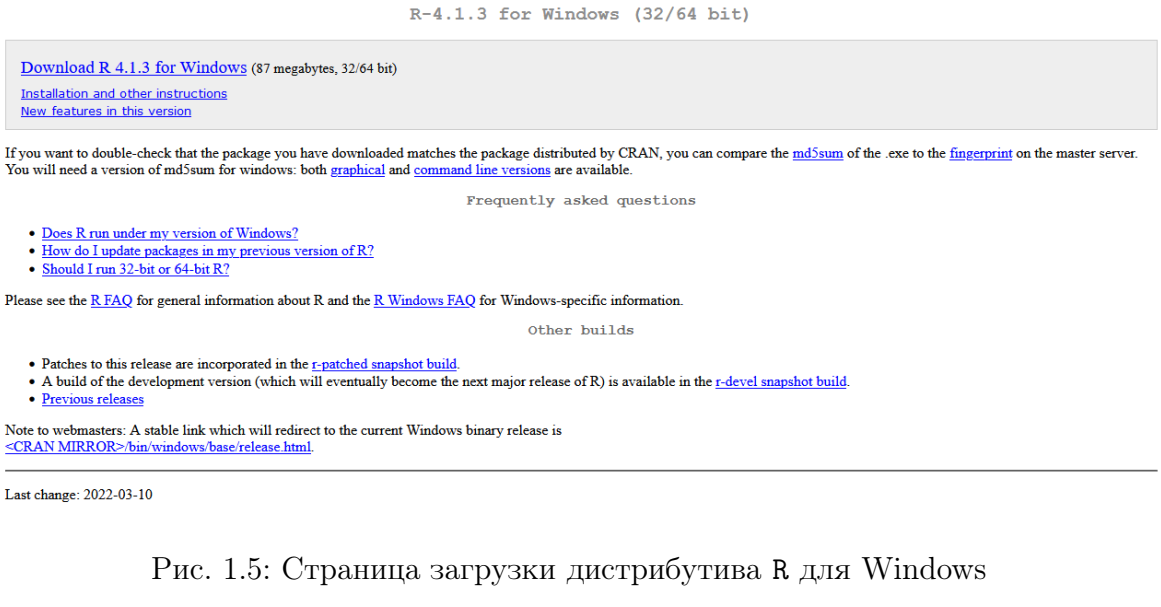
| | |
|---|---|
| base | Binaries for base distribution. This is what you want to **install R for the first time**. |
| contrib | Binaries of contributed CRAN packages (for R >= 3.4.x). |
| old contrib | Binaries of contributed CRAN packages for outdated versions of R (for R < 3.4.x). |
| Rtools | Tools to build R and R packages. This is what you want to build your own packages on Windows, or to build R itself. |

Please do not submit binaries to CRAN. Package developers might want to contact Uwe Ligges directly in case of questions / suggestions related to Windows binaries.

You may also want to read the R FAQ and R for Windows FAQ.

Note: CRAN does some checks on these binaries for viruses, but cannot give guarantees. Use the normal precautions with downloaded executables.

Рис. 1.4: Страница загрузки R для Windows



R-4.1.3 for Windows (32/64 bit)

Download R 4.1.3 for Windows (87 megabytes, 32/64 bit)
Installation and other instructions
New features in this version

If you want to double-check that the package you have downloaded matches the package distributed by CRAN, you can compare the md5sum of the .exe to the fingerprint on the master server. You will need a version of md5sum for windows: both graphical and command line versions are available.

Frequently asked questions

- Does R run under my version of Windows?
- How do I update packages in my previous version of R?
- Should I run 32-bit or 64-bit R?

Please see the R FAQ for general information about R and the R Windows FAQ for Windows-specific information.

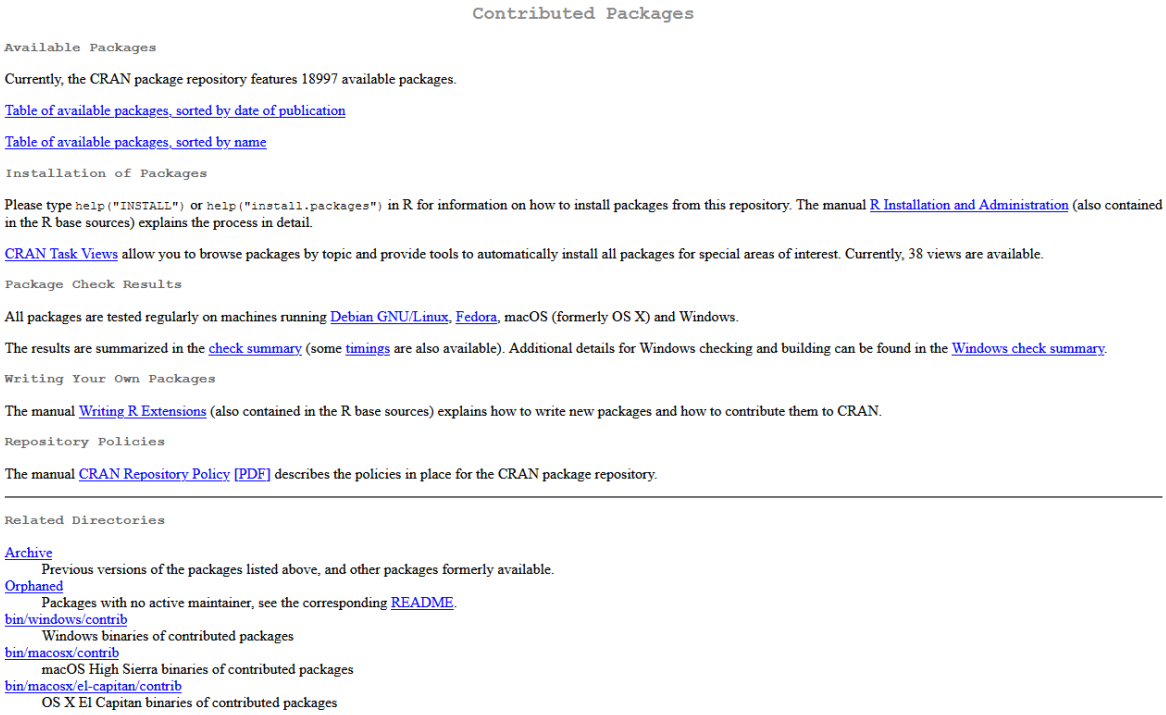
Other builds

- Patches to this release are incorporated in the r-patched snapshot build.
- A build of the development version (which will eventually become the next major release of R) is available in the r-devel snapshot build.
- Previous releases

Note to webmasters: A stable link which will redirect to the current Windows binary release is <CRAN MIRROR>/bin/windows/base/release.html.

Last change: 2022-03-10

Рис. 1.5: Страница загрузки дистрибутива R для Windows



Contributed Packages

Available Packages

Currently, the CRAN package repository features 18997 available packages.

Table of available packages, sorted by date of publication

Table of available packages, sorted by name

Installation of Packages

Please type help("INSTALL") or help("install.packages") in R for information on how to install packages from this repository. The manual R Installation and Administration (also contained in the R base sources) explains the process in detail.

CRAN Task Views allow you to browse packages by topic and provide tools to automatically install all packages for special areas of interest. Currently, 38 views are available.

Package Check Results

All packages are tested regularly on machines running Debian GNU/Linux, Fedora, macOS (formerly OS X) and Windows.

The results are summarized in the check summary (some timings are also available). Additional details for Windows checking and building can be found in the Windows check summary.

Writing Your Own Packages

The manual Writing R Extensions (also contained in the R base sources) explains how to write new packages and how to contribute them to CRAN.

Repository Policies

The manual CRAN Repository Policy [PDF] describes the policies in place for the CRAN package repository.

Related Directories

Archive
Previous versions of the packages listed above, and other packages formerly available.
Orphaned
Packages with no active maintainer, see the corresponding README.
bin/windows/contrib
Windows binaries of contributed packages
bin/macosx/contrib
macOS High Sierra binaries of contributed packages
bin/macosx/el-capitan/contrib
OS X El Capitan binaries of contributed packages

Рис. 1.6: Пакеты расширения, созданные сообществом

в виде соответствующего пакета расширения. Поэтому всё, что нужно — найти этот пакет, загрузить и установить, разобраться и использовать.

Иногда существует несколько способов решения задачи. В этом случае требуется выбрать один вариант из нескольких.

На странице `CRAN`, в нижней части раздела «Source Code for all Platforms» (исходный код для всех платформ), имеется пункт «Contributed extension packages» (рис. 1.6).

Так как имеется *огромное* количество всевозможных, потенциально полезных, пакетов, то для облегчения поиска можно использовать списки названий, отсортированные по дате публикации пакета или по его имени.

Более удобным является обзор пакетов, упорядоченных по решаемым задачам: CRAN Task Views (рис. 1.7).

Установка пакета расширения выполняется командой `install.packages`. Другой способ: выбрать пункт `Установить пакет(ы)...` в меню `Пакеты` графической оболочки `R`.

Для обновления установленных пакетов можно использовать команду `update.packages` или пункт `Обновить пакеты...` в меню `Пакеты` графической оболочки `R`.

*Замечание.* При администрировании пакетов расширения (установке, обновлении и/или удалении[1]) необходимо иметь права администратора.

[1] Команда `remove.packages`.

## CRAN Task Views

CRAN task views aim to provide some guidance which packages on CRAN are relevant for tasks related to a certain topic. They give a brief overview of the included packages and can be automatically installed using the ctv package. The views are intended to have a sharp focus so that it is sufficiently clear which packages should be included (or excluded) - and they are *not* meant to endorse the "best" packages for a given task.

- To automatically install the views, the ctv package needs to be installed, e.g., via
  `install.packages("ctv")`
  and then the views can be installed via `install.views` or `update.views` (where the latter only installs those packages are not installed and up-to-date), e.g.,
  `ctv::install.views("Econometrics")`
  `ctv::update.views("Econometrics")`
- The task views are maintained by volunteers. You can help them by suggesting packages that should be included in their task views. The contact e-mail addresses are listed on the individual task view pages.
- For general concerns regarding task views contact the ctv package maintainer.

### Topics

| | |
|---|---|
| Bayesian | Bayesian Inference |
| ChemPhys | Chemometrics and Computational Physics |
| ClinicalTrials | Clinical Trial Design, Monitoring, and Analysis |
| Cluster | Cluster Analysis & Finite Mixture Models |
| Databases | Databases with R |
| DifferentialEquations | Differential Equations |
| Distributions | Probability Distributions |
| Econometrics | Econometrics |
| Environmetrics | Analysis of Ecological and Environmental Data |
| ExperimentalDesign | Design of Experiments (DoE) & Analysis of Experimental Data |
| ExtremeValue | Extreme Value Analysis |
| Finance | Empirical Finance |
| FunctionalData | Functional Data Analysis |
| Genetics | Statistical Genetics |
| GraphicalModels | Graphical Models |
| HighPerformanceComputing | High-Performance and Parallel Computing with R |
| Hydrology | Hydrological Data and Modeling |
| MachineLearning | Machine Learning & Statistical Learning |
| MedicalImaging | Medical Image Analysis |
| MetaAnalysis | Meta-Analysis |
| MissingData | Missing Data |
| ModelDeployment | Model Deployment with R |
| NaturalLanguageProcessing | Natural Language Processing |
| NumericalMathematics | Numerical Mathematics |
| OfficialStatistics | Official Statistics & Survey Statistics |
| Optimization | Optimization and Mathematical Programming |
| Pharmacokinetics | Analysis of Pharmacokinetic Data |
| Phylogenetics | Phylogenetics, Especially Comparative Methods |
| Psychometrics | Psychometric Models and Methods |
| ReproducibleResearch | Reproducible Research |
| Robust | Robust Statistical Methods |
| Spatial | Analysis of Spatial Data |
| SpatioTemporal | Handling and Analyzing Spatio-Temporal Data |
| Survival | Survival Analysis |
| TeachingStatistics | Teaching Statistics |
| TimeSeries | Time Series Analysis |
| Tracking | Processing and Analysis of Tracking Data |
| WebTechnologies | Web Technologies and Services |

Рис. 1.7: Тематический каталог пакетов расширений

# Глава 2

# Основы R

Некоторые полезные команды:

- `?команда` или `help(команда)` — справка по команде.
- `??слово` — в справочной системе выполняется нечёткий поиск разделов, содержащих данное слово.
- `library(библиотека)` — загрузка библиотеки расширения.
- `library(help='библиотека')` — справка по библиотеке.
- `?библиотека:команда` — справка по команде из указанной библиотеки.
- `ls()` — вывести список объектов, содержащихся в рабочей памяти.
- `rm(объект)` — удалить объект.
- `rm(list=ls())` — очистить рабочую память.
- `q()` или `quit()` — завершение текущей сессии R.

`Ctrl+L` — очистить рабочее окно (при работе в графической оболочке `Rgui` и в консоли).

В этой главе и далее при описании сигнатуры функции часто будет использоваться сокращённый вариант, при котором будут опускаться редко используемые необязательные параметры. Для поиска более полной информации используйте `?`, `??` или `help`.

## 2.1 Основные команды

Начнём с арифметики: вычислить значение простого выражения не просто, а очень просто. Предположим, например, что требуется узнать значение выражения

$$1 + 2 - \frac{\sqrt{3 \times 4}}{5^6}.$$

После ввода в командной строке

```
1+2-(3*4)^0.5/5^6
```

и нажатия на клавишу Enter , ↵ или `Return`, в зависимости от клавиатуры, на экран будет выведено:

```
[1] 2.999778
```

Здесь [1] означает номер возвращаемого элемента (полезно, если возвращается длинный вектор), а 2.999778 — собственно вычисленный результат.

Команды-стрелки <- или -> используются для записи значения в переменную и позволяют указать на то, *что* является значением, а *что* — переменной, в которую это значение следует записать. Таким образом,

```
x <- 1+2-3*4/5^6
```

и

```
1+2-3*4/5^6 -> x
```

дают один и тот же результат: значение выражения будет записано в переменную x. Для присваивания переменной x значения value также можно использовать команду assign:

```
assign(x, value)
```

Для вывода содержимого переменной x на экран следует ввести имя переменной и нажать на Enter:

```
x
```

### 2.1.1 Арифметика

Переменная .Machine содержит информацию о представлении чисел на компьютере, на котором выполняется R. Команда names выдаёт список имён элементов или полей, содержащихся в объекте.

```
names(.Machine)
```

```
 [1] "double.eps"                "double.neg.eps"
 [3] "double.xmin"               "double.xmax"
 [5] "double.base"               "double.digits"
 [7] "double.rounding"           "double.guard"
 [9] "double.ulp.digits"         "double.neg.ulp.digits"
[11] "double.exponent"           "double.min.exp"
[13] "double.max.exp"            "integer.max"
[15] "sizeof.long"               "sizeof.longlong"
[17] "sizeof.longdouble"         "sizeof.pointer"
[19] "longdouble.eps"            "longdouble.neg.eps"
[21] "longdouble.digits"         "longdouble.rounding"
[23] "longdouble.guard"          "longdouble.ulp.digits"
[25] "longdouble.neg.ulp.digits" "longdouble.exponent"
[27] "longdouble.min.exp"        "longdouble.max.exp"
```

Для доступа к определённому полю объекта используется знак доллара $:

```
.Machine$double.eps    # 2.220446e-16
```

#### Целые числа

Для создания вектора указанной длины из целых чисел[1] (нулей) применяется команда integer:

```
integer(length = 0)
```

[1] В R величины типа integer представляют собой 32-битовые целые числа.

Так как по умолчанию параметр `length` равен нулю, то требуется явно указать, сколько целых чисел должно быть создано.

```
integer()+1      # numeric(0)
integer(1)+1     # 1
integer(10)+1    # 1 1 1 1 1 1 1 1 1 1
```

Функция `as.integer`

```
as.integer(x)
```

пытается преобразовать `x` в целые числа (аналогично `trunc`):

```
as.integer(c(1, 2.5, '256', '0xFF'))  # 1 2 256 255
```

Для проверки, имеет ли $x$ тип `integer`, используется функция `is.integer`:

```
is.integer(x)
```

```
is.integer(1)                    # внезапно, нет!
is.integer(as.integer(1))        # да
```

Унарные и бинарные арифметические операторы, используемые в `R`:

```
x <- as.integer(13)
y <- as.integer(4)

+x          # унарный плюс
-x          # унарный минус
x + y
x - y
x * y
x / y
x %/% y     # целочисленное деление
x %% y      # остаток от целочисленного деления
x ^ y       # возведение в степень
```

По определению, результат `1^y` и `y^0` *всегда* 1. Также

```
1/0         #  Inf положительная бесконечность
-1/0        # -Inf отрицательная бесконечность
0/0         #  NaN не число
```

Так как $(-2)^3 = -8$, то, очевидно, $\sqrt[3]{-8} = -2$. Однако

```
(-8)^(1/3)   # NaN: результат возведения отрицательного числа в дробную степень
```

### Числа с плавающей точкой

Для создания вектора указанной длины из чисел с плавающей точкой[2] — нулей, используется функция `double`:

```
double(length = 0)
```

`R` *всегда* использует для дробных значений представление в виде чисел с плавающей точкой двойной точности (double). Арифметические действия с такими величинами выполняются в соответствии со стандартом IEC 60559.

Функция `as.double` :

[2] В России для отделения целой части числа от дробной используется запятая, но в большинстве языков программирования и заграничных программ (кроме таких, как Microsoft Excel, использующих национальные настройки) применяется разделяющая *точка*.

```
as.double(x)
```

пытается преобразовать `x` в чисела с плавающей точкой (аналогично `as.numeric`):

```
as.double(c(1, 2.5, '123.456', '0xFF'))  # 1.000  2.500 123.456 255.000
as.double(c(NA, NaN, Inf, '-infinity'))  # NA  NaN  Inf -Inf
```

Для проверки, имеет ли $x$ тип `double`, используется функция `is.double`:

```
is.double(x)
```

```
is.double(1)                    # TRUE
is.double(NaN)                  # TRUE
is.double(Inf)                  # TRUE
is.double(NA)                   # FALSE
is.double(as.double(NA))        # TRUE
```

Для преобразования $\mathbb{R} \to \mathbb{Z}$ используются функции `ceiling` (округление вверх), `floor` (округление вниз), `trunc` (отбрасывание дробной части), а также `round` (округление до заданного количества цифр после точки по правилам математики):

```
ceiling(x)
floor(x)
trunc(x)
round(x, digits = 0)
```

```
x <- c(1.75,-1.75)
ceiling(x)          # 2 -1
floor(x)            # 1 -2
trunc(x)            # 1 -1

x <- c(-1.23449,-1.2345,-1.23451,1.23449,1.2345,1.23451)
round(x, digits=3) # -1.234 -1.234 -1.235 1.234 1.234 1.235
```

В особых случаях можно использовать зарезервированные слова `R`: `Inf` и `NaN`. `Inf` и `-Inf` обозначают положительную и отрицательную бесконечности, а `NaN` имеет значение «не число» (Not a Number). Все они могут использоваться как числовые значения, а также как вещественные и мнимые части комплексных чисел, но не как целые числа.

Для проверки, является объект конечным числом, бесконечной величиной или «не числом» используются функции `is.finite`, `is.infinite` и `is.nan` соответственно.

```
is.finite(x)
is.infinite(x)
is.nan(x)
```

```
x <- c(1/1, 1/0, 0/0)

is.finite(x)     # TRUE  FALSE FALSE
is.infinite(x)   # FALSE TRUE  FALSE
is.nan(x)        # FALSE FALSE TRUE
```

### Комплексные числа

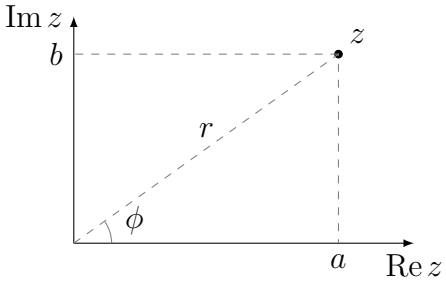
Не существует вещественного числа, квадрат которого был бы равен $-1$:

```
(-1)^0.5          # NaN
```

Но если перейти в поле[3] комплексных чисел $\mathbb{C}$, то результатом

```
(-1+0i)^0.5       # 0+1i
```

будет мнимая единица. Комплексное число $z \in \mathbb{C}$ можно представить в виде точки на комплексной плоскости $(\mathrm{Re}, \mathrm{Im})$,




положение которой можно записать, используя декартовы координаты $a$ (вещественная часть) и $b$ (мнимая часть):

$$z = a + bi$$

или полярные координаты $r$ (абсолютная величина, модуль) и $\phi$ (аргумент):

$$z = re^{\phi i}.$$

При этом

$$a = r\cos(\phi), \qquad r = \sqrt{a^2 + b^2},$$
$$b = r\sin(\phi), \qquad \phi = \operatorname{arctg}\frac{b}{a}.$$

При создании комплексных чисел, особенно при программировании, можно использовать функцию `complex`:

```
complex(length.out = 0, real = numeric(), imaginary = numeric(),
        modulus = 1, argument = 0)
```

```
complex(2)                                  # 0 + 0i, 0 + 0i
complex(real=1:3,imaginary=4:6)             # 1 + 4i, 2 + 5i, 3 + 6i
complex(modulus=1,argument=c(0,pi/4,pi/2))  # 1, √0.5 + √0.5i, 1i
```

Стандартные функции для работы с комплексными числами:

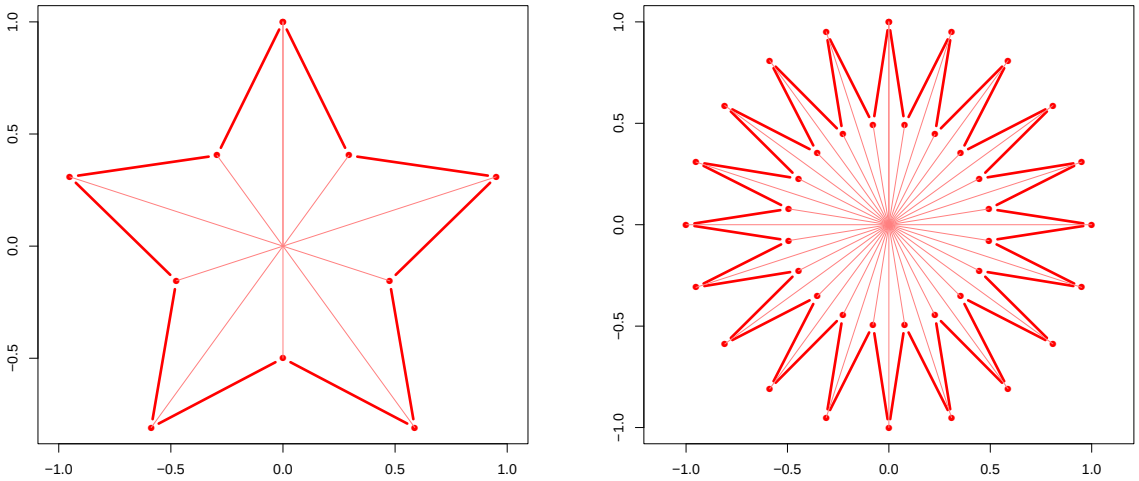
```
z <- complex(real=3,imag=4)  # x = 3 + 4i
Re(z)        # вещественная часть: a = 3
Im(z)        # мнимая часть: b = 4
Mod(z)       # модуль: r = √(3^2 + 4^2)
Arg(z)       # аргумент: ϕ = arctg(4/3)
Conj(z)      # комплексно-сопряжённое число: 3 − 4i
```

[3] *Полем* называется множество $X$, в котором определены две операции, каждая из которых сопоставляет двум элементам множества $X$ третий элемент из этого же множества. См., например, [23].

**Пример.** *Использование комплексных чисел для рисования звезды с n лучами.*

```
n <- 5   # число лучей

p <- complex(modulus=rep(c(1,0.5),n),
             argument=seq(from=pi/2,by=pi/n,length=2*n))
ind <- c(1:(2*n),1)

plot(Re(p[ind]),Im(p[ind]),type='b',xlab='',ylab='',
     lwd=3,col='red',pch=16,asp=1)
for (i in ind) lines(c(0,Re(p[i])), c(0,Im(p[i])), col=rgb(1,.5,.5))
```

Звёзды с 5-ю и 20-ю лучами:




❑

### 2.1.2 Векторы

Вектор — основная структура данных, используемая в R. Векторы могут использоваться как сами по себе, так и служить составными частями более сложных структур.

**Пример.** *Создание вектора, содержащего целые числа от* 1 *до* 10.

Такой вектор (в данном случае — вектор-столбец) можно создать несколькими способами.

- Функция c объединяет аргументы в вектор-столбец:

  ```
  c(1,2,3,4,5,6,7,8,9,10)
  ```

- Другой вариант — использовать конструкцию $n_1{:}n_2$, которая создаёт вектор $n_1, n_1 + 1, n_1 + 2, \ldots, n_2$:

  ```
  1:10
  ```

- Универсальный способ — применение генерирующей регулярные последовательности функции seq:

  ```
  seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
      length.out = NULL, along.with = NULL, ...)
  ```

Здесь

- `from` — начальное значение;
- `to` — конечное значение;
- `by` — длина шага;
- `length.out` — длина последовательности;
- `along.with` — длина соответствует длине параметра.

```
seq(from=1,to=10)              # от 1 до 10
seq(from=1,to=10,by=1)         # от 1 до 10 с шагом 1
seq(from=1,to=10,length=10)    # от 1 до 10, 10 элементов
seq(from=1,by=1,length=10)     # 10 элементов, начиная с 1, с шагом 1
seq(from=1,length=10)          # 10 элементов, начиная с 1
seq(to=10,length=10)           # 10 элементов, последний — 10
```

Длина шага по умолчанию — 1.

❑

Если требуется повторить число или последовательность несколько раз, то наш выбор — функция `rep`:

```
rep(1,5)          # 1,1,1,1,1
rep(c(1,2),5)     # 1,2,1,2,1,2,1,2,1,2
```

Также функция `rep` может использоваться, если требуется выделить место для хранения результатов вычислений. Другой способ выделения места в памяти — функция `double`.

Для функции `rep` существуют более быстрые специализированные версии: `rep.int` и `rep_len`:

```
rep.int(c(1,2,3),c(2,3,4))  # 1,1, 2,2,2, 3,3,3,3
rep_len(c(1,2,3),8)         # 1,2,3,1,2,3,1,2
```

Функция `length` возвращает количество элементов в векторе:

```
length(-10:10)   # 21
```

Для доступа к элементам вектора используются прямоугольные скобки `[ ]`: Пусть $x$ — вектор, содержащий квадраты целых чисел от 1 до 10:

```
x <- (1:10)^2
```

Тогда

```
x                # все элементы вектора
x[]              # аналогично x
x[3]             # третий элемент вектора
x[-3]            # все элементы вектора за исключением третьего
x[3:7]           # элементы x3,...,x7
x[-7:-3]         # все элементы вектора за исключением x3,...,x7
x[c(1,3,8,10)]   # элементы x1,x3,x8,x10
x[x<=5]          # все элементы вектора, не превосходящие 5
```

Для вычисления суммы элементов вектора используется функция `sum`, для произведения — `prod`:

```
x <- 1:10
sum(x)          # 55
prod(x)         # 3628800
```

Функции `max` и `min` ищут наибольший и наименьший элемент вектора соответственно:

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

min(x)  # -2
max(x)  # 3
```

Функция `which` применяется для поиска индексов элементов, соответствующих заданному логическому значению:

```
which(x, arr.ind = FALSE, useNames = TRUE)

which(x==min(x))  # индексы минимальных элементов
which(x==max(x))  # индексы максимальных элементов
which(x%%2==0)    # индексы чётных элементов
which(x%%2!=0)    # индексы нечётных элементов
```

Функции `cumsum`, `cumprod`, `cummax` и `cummin` применяются для вычисления сумм, произведений, максимумов и минимумов с *накоплением*:

```
x <- 1:10

cumsum(x)       # 1, 1+2, 1+2+3, 1+2+3+4, ...
cumprod(x)      # 1, 1×2, 1×2×3, 1×2×3×4, ...
cummax(x)       # 1, 2, 3, 4, ...
cummin(x)       # 1, 1, 1, 1, ...
```

### 2.1.3 Строки

Функция `character` используется для создания `length` вектора пустых строк.

```
character(length = 0)
```

Для проверки, является ли `x` строкой, используется функция `is.character`:

```
is.character(x)
```

**Преобразование в строку** выполняется функцией `as.character`:

```
as.character(x, ...)
```

Также для преобразования невычисленных выражений в строку может применяться функция `deparse` и её более простой вариант `deparse1`:

```
deparse(expr, width.cutoff = 60L,
        backtick = mode(expr) %in% c('call', 'expression', '(', 'function'),
        control = c('keepNA', 'keepInteger', 'niceNames', 'showAttributes'),
        nlines = -1L)

deparse1(expr, collapse = ' ', width.cutoff = 500L, ...)
```

где

- `expr` — преобразуемое в строку выражение.
- `width.cutoff` — целое число (в диапазоне от 20 до 500), определяющее место разреза (в байтах) строки-результата.
- `backtick` — логическое выражение, определяющее, следует ли окружать символьные имена, не удовлетворяющие стандартному синтаксису, обратными апострофами.
- `control` — вектор для настройки поведения функции.
- `nlines` — максимальное количество строк в результате. Отрицательное число означает отсутствие ограничений.
- `collapse` — символ, используемый при конкатенации строк.

```
deparse(1)                      # 1
deparse(sin(1))                 # 0.841470984807897
deparse(c(1,-2,NaN))            # c(1, -2, NaN)
deparse(list(x=1,y=sin(1)))     # list(x = 1, y = 0.841470984807897)
deparse(Inf)                    # Inf
deparse(NaN)                    # NaN
```

**Выделение подстроки** выполняется функциями `substr` и `substring`, которые также могут использоваться и для замены подстроки на `value`:

```
substr(x, start, stop)
substring(text, first, last = 1000000L)
substr(x, start, stop) <- value
substring(text, first, last = 1000000L) <- value
```

```
x <- 'Очень␣длинная␣строка'
substr(x,7,13)                          # длинная
substr(x,7,13) <- 'короткая'; x         # Очень коротка строка
substring(x,7) <- 'короткая␣строка'; x  # Очень короткая строка
```

**Разбиение на подстроки** выполняется функцией `strsplit`:

```
strsplit(x, split, fixed = FALSE, perl = FALSE, useBytes = FALSE)
```

Здесь

- `x` — вектор, содержащий строки, которые требуется разделить на подстроки;
- `split` — строка, содержащая шаблоны, по которым будет выполняться разбиение;
- `fixed` — если `TRUE`, то выполняется явное сравнение со `split`. При `FALSE` в `split` могут использоваться шаблоны (имеет больший приоритет по сравнению с `perl`);
- `perl` — следует ли использовать регулярные выражения в стиле языка Perl;
- `useBytes` — использовать ли побайтовое сравнение с шаблоном.

**Пример.** *Разбиение двух строк на слова.*

```
strsplit(c('короткая␣строка','более␣длинная␣строка'),'␣')
```

```
[[1]]
[1] "короткая" "строка"

[[2]]
[1] "более"   "длинная" "строка"
```

❑

**Пример.** *Разбиение предложений на слова.*

```
s1 <- 'короткая строка. более длинная строка.'; s1
```

```
[1] "короткая строка. более длинная строка."
```

Разбиваем по отдельным предложениям:

```
s2 <- strsplit(s1,'[[:punct:]]'); s2
```

```
[[1]]
[1] "короткая строка"       " более длинная строка"
```

Удаляем пробелы в началах и концах строк с помощью функции `trimws`:

```
s3 <- sapply(s2,trimws); s3
```

```
     [,1]
[1,] "короткая строка"
[2,] "более длинная строка"
```

Разбиваем предложения на слова:

```
strsplit(s3,'[[:space:]]')
```

```
[[1]]
[1] "короткая" "строка"

[[2]]
[1] "более"   "длинная" "строка"
```

❑

**Регулярные выражения** стандарта POSIX 1003.2

- Синтаксис:
  - `.` — любой одиночный символ, внутри квадратных скобок обозначает точку;
  - `[ ]` — множество одиночных символов, можно также использовать диапазоны:
    `[abc]` — символы `a`, `b` и `c`;
    `[a-z]` — символы `a`, `b`, . . . , `z`;
  - `[^ ]` — множество одиночных символов, которые исключаются из сравнения;
  - `^` — начало строки;
  - `$` — конец строки;

  - `*` — предшествующий элемент отсутствует или встречается любое число раз;
  - `?` — предшествующий элемент отсутствует или встречается один раз;
  - `+` — предшествующий элемент встречается один или большее число раз;
  - `{m}` — предшествующий элемент должен встретиться в точности $m$ раз;
  - `{m,}` — предшествующий элемент должен встретиться как минимум $m$ раз;
  - `{m,n}` — предшествующий элемент должен встретиться не меньше $m$ и не больше $n$ раз;
  - `( )` — подвыражение.

- Классы символов:
  - `[:upper:]` (аналогично `[A-Z]`) — прописные буквы;
  - `[:lower:]` (аналогично `[a-z]`) — строчные буквы;
  - `[:alpha:]` (аналогично `[A-Za-z]`) — буквы;
  - `[:digit:]` (аналогично `[0-9]`) — цифры;
  - `[:xdigit:]` (аналогично `[0-9A-Fa-f]`) — цифры в 16-ричной системе счисления;
  - `[:alnum:]` (аналогично `[0-9A-Za-z]`) — цифры и буквы;
  - `[:punct:]` — символы пунктуации:

    `! " # % & ' ( ) * + - , . / : ;  < = > ? @ [ \ ] ^ _ ‘ { | } ~`;
  - `[:blank:]` (аналогично `[ \t]`) — пробел и символ табуляции;
  - `[:space:]` (аналогично `[ \t\n\r\f\v]`) — пробельные символы;
  - `[:cntrl:]` — управляющие символы;
  - `[:graph:]` (аналогично `[[:alnum:][:punkt:]]`) — все символы, имеющие графическое представление;
  - `[:print:]` (аналогично `[ [:graph:]]`) — все символы, имеющие графическое представление, и пробел;
  - `[:word:]` — цифры и буквы, а также символ подчёркивания.

**Определение длин строк** выполняется функциями `nchar` и `nzchar`.

```
nchar(x, type = "chars", allowNA = FALSE, keepNA = NA)
nzchar(x, keepNA = FALSE)
```

- `x` — вектор строк, длины которых вычисляются.
- `type` — способ вычисления длины:
  - `bytes` — число байт, необходимое для хранения строки;
  - `chars` — количество символов;
  - `width` — количество столбцов, используемых командой `cat` для печати моноширинным шрифтом.

```
s <- unlist(strsplit('лев␣николаевич␣толстой','␣'))
nchar(s)                # 3      10        7
```

**Конкатенация строк** выполняется с помощью функций paste и paste0:

```
paste (..., sep = ' ', collapse = NULL, recycle0 = FALSE)
paste0(...,            collapse = NULL, recycle0 = FALSE)
```

```
paste('one','two',sep='␣')           # one two
paste('one','two','three',sep='␣')   # one two three
```

**Пример.** *Функция для объединения строк-элементов вектора.*

```
concat <- function(ss,sep='␣')
{
  concat_iter <- function(w,ss,s)
    if (length(ss)==0)
      paste(s,w,sep=sep)
    else
      concat_iter(ss[1],ss[-1],paste(s,w,sep=sep))

  concat_iter(character(0),ss,character(0))
}
```

Использование concat для удаления лишних пробелов[4]:

```
s <- "Oh,␣␣␣␣␣be␣␣␣␣a␣␣fine␣␣␣␣␣girl,␣␣␣␣kiss␣␣␣me."
# разбиваем строку на слова
ss <- unlist(strsplit(s,'␣'))
ss
```

```
 [1] "Oh,"   ""      ""      ""      ""      "be"    ""      ""      "a"
[10] ""      "fine"  ""      ""      ""      "girl," ""      ""      ""
[19] "kiss"  ""      ""      "me."
```

Для удаления пробелов в начале и в конце строки применяется функция trimws.

```
# объединяем слова ненулевой длины и удаляем пробелы на границах
trimws(concat(ss[nchar(ss)>0]))
```

```
[1] "Oh, be a fine girl, kiss me."
```

❑

**Форматирование строки** выполняется функцией format:

```
format(x, ...)
## Default S3 method:
format(x, trim = FALSE, digits = NULL, nsmall = 0L,
       justify = c("left", "right", "centre", "none"),
       width = NULL, na.encode = TRUE, scientific = NA,
       big.mark   = "",   big.interval = 3L,
       small.mark = "", small.interval = 5L,
       decimal.mark = getOption("OutDec"),
       zero.print = NULL, drop0trailing = FALSE, ...)
```

[4]Выражение "Oh, be a fine girl, kiss me" традиционно используется людьми, знающими английский язык, как мнемоническая последовательность для запоминания спектральных классов звёзд от самых горячих "O" до самых холодных "M".

Женщины также могут использовать "Oh, be a fine guy, kiss me".

- `x` — любой объект `R`, обычно число.
- `trim` — подавлять ли начальные пробелы, используемые для выравнивания вправо логических, числовых и комплексных величин.
- `digits` — количество значащих цифр в числовых и комплексных величинах.
- `nsmall` — минимальное количество цифр дробной части в числовых и комплексных величинах. Возможны значения от 0 до 20 включительно.
- `justify` — способ выравнивания.
- `width` — минимальная ширина поля. Если `NULL` или `0`, то ширина не ограничена.

**Вывод строки** проще всего выполнить с помощью функции `cat`:

```
cat(... , file = "", sep = " ", fill = FALSE, labels = NULL, append = FALSE)
```

```
cat(format('x',width=10,justify='centre'),
    format('sin(x)',width=15,justify='centre'),
    '\n')
```

```
    x          sin(x)
```

```
for (x in seq(from=0,to=pi/2,length=10))
  cat(format(x,width=10,justify='right'),
      format(sin(x),justify='left',digits=10,width=15),
      '\n')
```

```
         0              0
 0.1745329    0.1736481777
.........................
  1.396263     0.984807753
  1.570796              1
```

Функция `print` имеет больше возможностей по сравнению с `cat`:

```
print(x, ...)
## S3 method for class 'factor'
print(x, quote = FALSE, max.levels = NULL,
      width = getOption("width"), ...)
## S3 method for class 'table'
print(x, digits = getOption("digits"), quote = FALSE,
      na.print = "", zero.print = "0",
      right = is.numeric(x) || is.complex(x),
      justify = "none", ...)
## S3 method for class 'function'
print(x, useSource = TRUE, ...)
```

**Кодирование** строки (перевод из одной кодировки в другую) осуществляется функцией iconv, которая использует для этого возможности операционной системы:

```
iconv(x, from = '', to = '', sub = NA, mark = TRUE, toRaw = FALSE)
```

- sub — строка символов, для которых будет выполняться перекодирование.

```
s <- 'Раз␣два␣три.␣One␣two␣three.'
iconv(s,to='windows-1251')                          # Раз два три. One two three.
iconv(s,to='UTF-8')                                 # Раз два три. One two three.
iconv(s,to='UTF-8',toRaw=TRUE)                      # d0 a0 d0 b0 d0 b7 ...
iconv(s,to='866')                                   # Ђ § ЋЎ  ваЁ. One two three.
iconv(iconv(s,to='866'),from='866',to='UTF-8')      # Раз два три. One two three.
```

Все платформы R поддерживают кодировки '' (текущая кодовая страница), latin1 и UTF-8. Функция iconvlist выдаёт список доступных кодировок:

```
iconvlist()
```

```
  [1] "437"                  "850"                  "852"
  [4] "855"                  "857"                  "860"
  [7] "861"                  "862"                  "863"
 [10] "865"                  "866"                  "869"
.....................................................................
 [31] "CP12000"             "CP12001"              "CP1201"
 [34] "CP1250"              "CP1251"               "CP1252"
.....................................................................
[286] "UCS-2BE"             "UCS-2LE"              "UCS-4"
[289] "UCS-4BE"             "UCS-4BE"              "UCS-4LE"
[292] "UCS-4LE"             "UCS2"                 "UCS2BE"
[295] "UCS2LE"              "UCS4"                 "UCS4BE"
[298] "UCS4LE"              "UHC"                  "unicodeFFFE"
[301] "US"                  "US-ASCII"             "UTF-16"
[304] "UTF-16BE"            "UTF-16LE"             "UTF-32"
[307] "UTF-32BE"            "UTF-32LE"             "UTF-8"
[310] "UTF16"               "UTF16BE"              "UTF16LE"
[313] "UTF32"               "UTF32BE"              "UTF32LE"
[316] "UTF8"                "WINBALTRIM"           "windows-1250"
[319] "windows-1251"        "windows-1252"         "windows-1253"
.....................................................................
```

### 2.1.4 Логические выражения

Для обозначения логических значений «истина» и «ложь» используются зарезервированные слова TRUE и FALSE соответственно. T и F — глобальные *переменные*, инициализированные значениями TRUE и FALSE:

```
TRUE; FALSE
T;    F
```

Поэлементное сравнение:

```
x <- 1:9          # 1,2,3,4,5,6,7,8,9
y <- -(-9:-1)     # 9,8,7,6,5,4,3,2,1

x == y   # x = y: FALSE FALSE FALSE FALSE TRUE  FALSE FALSE FALSE FALSE
x != y   # x ≠ y: TRUE  TRUE  TRUE  TRUE  FALSE TRUE  TRUE  TRUE  TRUE
x < y    # x < y: TRUE  TRUE  TRUE  TRUE  FALSE FALSE FALSE FALSE FALSE
x <= y   # x ⩽ y: TRUE  TRUE  TRUE  TRUE  TRUE  FALSE FALSE FALSE FALSE
x > y    # x > y: FALSE FALSE FALSE FALSE FALSE TRUE  TRUE  TRUE  TRUE
x >= y   # x ⩾ y: FALSE FALSE FALSE FALSE TRUE  TRUE  TRUE  TRUE  TRUE
```

Агрегирование результатов поэлементного сравнения выполняется с помощью функции `all` (все элементы истинны) и `any` (хотя бы один элемент истинен):

```
all(x==y)     # ∀i: x_i = y_i ? FALSE
any(x==y)     # ∃i: x_i = y_i ? TRUE
```

Объединение результатов:

```
!(x <= y)              # поэлементное отрицание
x <= y & x >= y        # поэлементная конъюнкция
x <= y && x >= y       # агрегированние конъюнкций слева направо
x <= y | x >= y        # поэлементная дизъюнкция
x <= y || x >= y       # агрегированние дизъюнкций слева направо
xor(x <= y, x >= y)    # поэлементное исключающее ИЛИ
```

Функции `isTRUE` и `isFALSE` проверяют, является ли аргумент скалярной логической величиной со значением `TRUE` или `FALSE` соответственно. Дополнительно выполняется проверка наличия значения аргумента (`!is.na`). Таким образом `isTRUE(x)` аналогично

```
is.logical(x) && length(x)==1 && !is.na(x) && x
```

а `isFALSE(x)` аналогично

```
is.logical(x) && length(x)==1 && !is.na(x) && !x
```

### 2.1.5 Встроенные константы

Заглавные и строчные латинские буквы, аббревиатуры и полные английские названия месяцев, а также первые несколько знаков числа $\pi$:

```
LETTERS       # A, B, C, ...
letters       # a, b, c, ...
month.abb     # Jan, Feb, Mar, ...
month.name    # January, February, March, ...
pi            # 3.141593
```

### 2.1.6 Встроенные функции

Для вычисления абсолютного значения и квадратного корня используются функции `abs` и `sqrt` соответственно:

```
abs(x)
sqrt(x)
```

*Замечание.* Вместо функции `sqrt` можно использовать возведение в степень 1/2 или 0.5.

```
abs(c(-1,0,1,1+1i,1-1i))     # 1.000000 0.000000 1.000000 1.414214 1.414214
abs(-2:2 < 0)                # 1 1 0 0 0
sqrt(c(-4,0,4))              # NaN 0 2
sqrt(as.complex(c(-4,0,4))) # 0+2i 0+0i 2+0i
as.complex(c(-4,0,4))^.5     # 0+2i 0+0i 2+0i
```

### Логарифм и показательная функция

```
x <- 25
log(x)           # натуральный логарифм: ln(x)
log(x, base=5)   # log_5(x)
log2(x)          # log_2(x)
log10(x)         # log_10(x)
log1p(x)         # ln(x+1)

log(-25)         # NaN
log(-25+0i)      # 3.218876 + 3.141593i

x <- 1
exp(x)           # экспонента
exp1p(x)         # exp(1+x)
```

### Тригонометрические функции

> *Товарищи курсанты, в военное время значение косинуса может достигать 2, а в исключительных случаях, когда того требует сложившаяся на фронтах обстановка, даже 3!*
>
> Анекдот (?)

```
x <- pi/6
cos(x)                  # cos(x)
sin(x)                  # cos(x)
tan(x)                  # cos(x)

x <- 1/6
cospi(x)                # cos(πx)
sinpi(x)                # sin(πx)
tanpi(x)                # tg(πx)
```

Обратные тригонометрические функции:

```
x <- 1/2
acos(x)                 # arccos(x)
asin(x)                 # arcsin(x)
atan(x)                 # arctg(x)
```

Функция `atan2(y,x)` возвращает угол между $x$-осью и вектором из начала координат в точку $(x, y)$ (поворот выполняется в положительном направлении — против часовой стрелки). Для положительных аргументов `atan2(y,x)` равен `atan(y/x)`.

```
atan2(1,1)*180/pi       # 45°
atan2(1,-1)*180/pi      # 135°
atan2(-1,1)*180/pi      # -45°
atan2(-1,-1)*180/pi     # -135°
```

*Замечание.* Значение синуса может быть больше единицы не только в военное время. Правда, для этого аргумент функции должен быть комплексным:

```
asin(2+0i)          # если x ≈ 1.570796 – 1.316958i
sin(asin(2+0i))     # то sin(x) = 2!
```

### Гиперболические функции

Конечно, и гиперболические, и обратные гиперболические функции можно реализовать самостоятельно.:

$$\operatorname{ch}(x) = \frac{e^x + e^{-x}}{2}, \quad \operatorname{sh}(x) = \frac{e^x - e^{-x}}{2}, \quad \operatorname{th}(x) = \frac{\operatorname{sh}(x)}{\operatorname{ch}(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

А можно использовать уже имеющиеся:

```
cosh(x)     # ch(x)
sihh(x)     # sh(x)
tanh(x)     # th(x)
```

Обратные гиперболические функции:

$$\operatorname{arch}(x) = \ln\left(x + \sqrt{x^2 - 1}\right), \quad \operatorname{arsh}(x) = \ln\left(x + \sqrt{x^2 + 1}\right), \quad \operatorname{arth}(x) = \frac{1}{2}\ln\left(\frac{x+1}{x-1}\right)$$

```
acosh(x)    # arch(x)
asihh(x)    # arsh(x)
atanh(x)    # arth(x)
```

### Специальные функции

Функции `gamma` и `beta` используются, например, при вычисленни значений функций распределения плотности вероятности случайных величин. Гамма-функция (см. [16: с. 81–84]):

`gamma(x)`

$$\Gamma(z) = \int_0^\infty s^{z-1}\exp(-s)\,ds, \quad \operatorname{Re} z > 0.$$

Бета-функция (см. [16: с. 84]):

`beta(a, b)`

$$\mathrm{B}(z, w) = \frac{\Gamma(z)\Gamma(w)}{\Gamma(z+w)}.$$

Функции `factorial` ($n!$ при $n \in \mathbb{N}_0$ или $\Gamma(x+1)$ для $n = x$ и $x \in \mathbb{R} \setminus \{-\mathbb{N}\}$) и `choose` (биномиальные коэффициенты) применяются, в частности, в комбинаторике:

`factorial(x)`

$$0! = 1, \quad n! = \prod_{i=1}^{n} i;$$

`choose(n, k)`

$$C_n^k = \binom{n}{k} = \frac{n!}{k!(n-k)!}.$$

## 2.2 Структуры данных

Описание каждого класса данных в R содержит

- `class` — тип объекта;
- `typeof` — тип данных объекта;
- `attributes` — метаданные: атрибуты объекта и их значения.

Например, для матрицы:

```
x <- matrix(1:15,nrow=3,ncol=5)
class(x)          # "matrix" "array"
typeof(x)         # "integer"
attributes(x)     # $dim 3 5
```

### 2.2.1 Матрицы

Матрица — прямоугольная таблица (возможно, в виде одной строки или одного столбца):

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

Здесь

- `data` — последовательность данных;
- `nrow` — количество строк;
- `ncol` — количество столбцов;
- `byrow` — формировать матрицу по столбцам (`FALSE`) или по строкам (`TRUE`);
- `dimnames` — если не `NULL`, то список, содержащий один (названия строк) или два (названия строк и столбцов соответственно) вектора.

По умолчанию матрица заполняется по столбцам:

```
matrix(1:6, nrow=2,ncol=3); A
```

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

Если размерность матрицы не позволяет вместить все элементы первого аргумента, то будет использована только часть:

```
matrix(1:6, nrow=2,ncol=2); A
```

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

Если элементов в первом аргументе не хватает для заполнения, то они будут повторены:

```
A <- matrix(1:6, nrow=3,ncol=4); A
```

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

Необязательный аргумент `byrow` позволяет указать, что матрицу следует заполнять по строкам:

```
A <- matrix(1:6, nrow=2,ncol=3, byrow=TRUE); A
```

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

Функция `length` возвращает количество элементов в матрице.

Функция `dim` возвращает двухэлементный вектор, содержащий число строк и столбцов матрицы.

Функции `nrow` и `ncol` возвращают число строк и число столбцов соответственно.

Для построения матрицы из векторов (вручную или в коде программы) используются функции `cbind` и `rbind`. Векторы-аргументы функцией `cbind` преобразуются в *столбцы* матрицы, а функцией `rbind` — в *строки*:

```
x <- 1:10
cbind(1,x,x^2) -> X    # X_ij = j^(i-1)
rbind(1,x,x^2) -> X    # X_ij = i^(j-1)
```

### Матричные операции

```
A <- matrix(c(1,1,1,1,2,3,1,4,9),nrow=3,ncol=3,byrow=TRUE)

det(A)                          # определитель

eigen(A)                        # собственные числа и собственные векторы
eigen(A,only.value=TRUE)        # только собственные числа

qr(A)                           # QR-разложение
svd(A)                          # сингулярное разложение
chol(t(A)%*%A)                  # разложение Холецкого
                                # (A — положительно определённая матрица)

solve(A)                        # обратная матрица A^(-1)
```

Описание приведённых выше матричных операций можно посмотреть, например, в [6] или [22].

**Пример** (Решение системы линейных алгебраических уравнений)**.** *Некий купец купил 100 рулонов красной и синей тканей, заплатив 6625 рублей*[5]*. Сколько куплено красной ткани и сколько синей, если цена одного рулона красной ткани — 75 рублей, а синей — 50 рублей?*

[5] Дело, похоже, происходило задолго до Великой Октябрьской социалистической революции. А может быть даже и до Февральской буржуазной. Рубль тогда был дорог...

Формально задачу можно записать в виде системы из двух линейных алгебраических уравнений:

$$\begin{aligned} \text{красная} + \text{синяя} &= 100 \\ 75 \cdot \text{красная} + 50 \cdot \text{синяя} &= 6625 \end{aligned}$$

или, с использованием матрично-векторной нотации:

$$Ax = b$$

где

$$A = \begin{pmatrix} 1 & 1 \\ 75 & 50 \end{pmatrix}, \quad b = \begin{pmatrix} 100 \\ 6625 \end{pmatrix}, \qquad x = \begin{pmatrix} \text{красная} \\ \text{синяя} \end{pmatrix}.$$

```
A <- matrix(c(1,1,75,50),nrow=2,ncol=2,byrow=TRUE,
            dimnames=list(c('','цена'),c('красная','синяя'))); A
```

```
     красная синяя
           1     1
цена      75    50
```

```
b <- matrix(c(100,6625),nrow=2,
            dimnames=list(c('метры','рубли'),c(''))); b
```

```
метры  100
рубли  6625
```

Для вычисления решения системы линейных алгебраических уравнений

$$x = A^{-1}b \tag{2.1}$$

используется функция `solve`:

```
solve(A,b)          # решение системы линейных алгебраических уравнений
```

```
красная 65
синяя   35
```

Тот же результат, но полученный менее эффективно с точки зрения использования вычислительных ресурсов, получается умножением обратной матрицы на вектор правой части (прямая реализация выражения (2.1)):

```
solve(A) %*% b      # обратная матрица умножается на вектор
```

❑

**Пример.** *Поиск* квазирешения *переопределённой системы линейных алгебраических уравнений:*

$$\begin{aligned} x_1 + x_2 &= 10, \\ 2x_1 - 3x_2 &= 5, \\ x_1 + 5x_2 &= 17. \end{aligned}$$

Так как система состоит из трёх уравнений с двумя неизвестными, то, вообще говоря, решения не существует — прямоугольная матрица не имеет обратной. Однако, если решения нет, но нам оно *очень* нужно, то можно заменить решение его суррогатом — квазирешением. Запишем исходную систему уравнений в матричной форме

$$Ax = b,$$

где

$$A = \begin{pmatrix} 1 & 1 \\ 2 & -3 \\ 1 & 5 \end{pmatrix}, \quad x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \text{ и } b = \begin{pmatrix} 10 \\ 5 \\ 17. \end{pmatrix}.$$

Добавим в правую часть невязку $\varepsilon$ и попробуем найти такое *квазирешение*, чтобы эта невязка была как можно ближе к нулю:

$$Ax = b + \varepsilon.$$

Чтобы получить систему линейных алгебраических уравнений с квадратной матрицей коэффициентов, применим левую трансформацию Гаусса, умножив обе части на $A^\top$:

$$A^\top Ax = A^\top b + A^\top \varepsilon.$$

Обозначим через $\tilde{x}$ такой вектор, при котором второе слагаемое в правой части обращается в ноль:

$$A^\top A\tilde{x} = A^\top b. \tag{2.2}$$

Тогда $\tilde{x}$ — искомое *квазирешение*[6]:

$$\tilde{x} = (A^\top A)^{-1} A^\top b. \tag{2.3}$$

```
A <- matrix(c(1,2,1, 1,-3,1),nrow=3,ncol=2)
b <- c(10,5,17)
solve(t(A)%*%A, t(A)%*%b)       # Реализация (2.2)
```

или

```
solve(t(A)%*%A) %*%t(A)%*%b     # Реализация (2.3)
```

```
     [,1]
[1,]  9.1
[2,]  4.4
```

*Замечание.* Выражения вида (2.3) часто используются в эконометрике при оценивании параметров линейных моделей. Но с точки зрения вычислительных затрат обращение матрицы с последующим умножением на вектор существенно проигрывает непосредственному решению системы уравнений (2.2).

Матрица $A^\top A$ — по построению симметричная (и, как мы надеемся, положительно определённая), что позволяет при использовании метода квадратных корней (см. [18: с. 165–168]) уменьшить объём вычислений при решении системы уравнений ещё наполовину.

*Замечание.* Поиск квазирешения переопределённой системы линейных алгебраических уравнений по сути сводится к использованию метода наименьших квадратов — подбору таких значений $\tilde{x}$, чтобы минимизировать сумму квадратов невязок:

$$\sum_i \varepsilon_i^2 \to \min.$$

[6] Матрица $A^+ = (A^\top A)^{-1} A^\top$ называется псевдообратной матрицей Мура-Пенроуза.

```
ESS <- function(x) sum((A%*%x-b)^2)
optim(c(1,2),ESS,method='BFGS')$par
```

```
[1] 9.099999 4.399999
```

❑

### 2.2.2 Массивы

Для создания (гипер)кубов данных — многомерных прямоугольных таблиц — применяется функция array:

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

Двумерный массив аналогичен матрице (см. раздел 2.2.1).

**Пример.** *Использование трёхмерного массива для хранения набора разноцветных точек.*

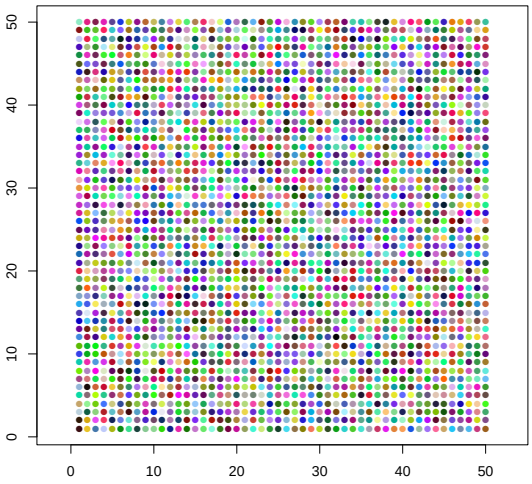
```
n <- 50
# n × n × 3 матрица M используется для хранения
# RGB-компонентов цветов n × n точек — узлов сетки
M <- array(runif(n*n*3),dim=c(n,n,3))
# RGB→цвет
vec2col <- function(x) rgb(x[1],x[2],x[3])
```

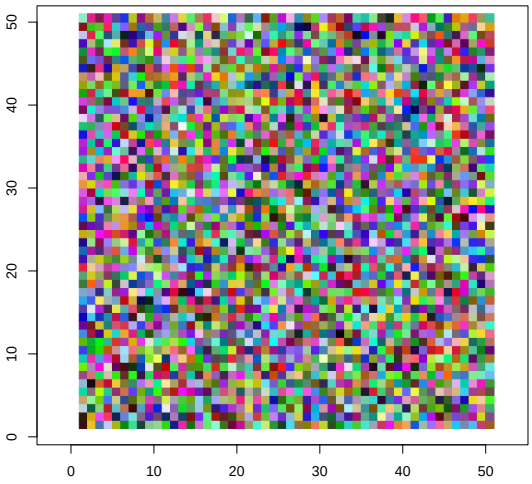
Рисование точками на чистом холсте:

```
# Чистый холст
plot(1:n,1:n,xlab='',ylab='',type='n',asp=1)
# Точки
for (i in 1:n)
for (j in 1:n)
  points(i,j,pch=16,col=vec2col(M[i,j,]))
```

Рисование прямоугольниками:

```
# Чистый холст
plot(1:n,1:n,xlab='',ylab='',type='n',asp=1)
# Прямоугольники
for (i in 1:n)
for (j in 1:n)
{
  color <- vec2col(M[i,j,])
  rect(i,j,i+1,j+1,col=color,border=color)
}
```




❑

### 2.2.3 Списки

Список может содержать именованные элементы различных типов:

`list(...)`

Аргументы создающей список функции `list` должны иметь вид `тэг = значение` или просто `значение`:

```
list(c(1,2,3),c('a','b','c','d'))

[[1]]
[1] 1 2 3

[[2]]
[1] "a" "b" "c" "d"

list(num=c(1,2,3), ch=c('a','b','c','d')) -> L
L

$num
[1] 1 2 3

$ch
[1] "a" "b" "c" "d"
```

Второй элемент списка:

```
L[2]

$num
[1] 1 2 3
```

Значение второго элемента списка:

```
L[[2]]
```

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

Значение элемента с тэгом `ch`:

```
L$ch
```

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

Количество элементов в списке:

```
length(L)
```

```
[1] 2
```

**Пример** (треугольник Паскаля). *Вычисление биномиальных коэффициентов* $C_n^m$.

$$\binom{n}{0} = \binom{n}{n} = 1, \quad n \in \mathbb{N}_0,$$
$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}, \quad 0 < k < n.$$

```
C <- list(c(1),c(1,1))
for (i in 2:9)
  C[[i+1]] <- c(C[[i]][1],C[[i]][-1]+C[[i]][-i],C[[i]][i])

for (i in 1:length(C)) cat(C[[i]],'\n')
```

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
1 10 45 120 210 252 210 120 45 10 1
```

❑

Команда `unlist` пытается преобразовать список в вектор:

```
unlist(x, recursive = TRUE, use.names = TRUE)
```

Если `recursive = FALSE`, то одноуровневый список преобразуется в вектор, а многоуровневые списки не изменяются:

```
L <- list(f=c(1,2),s=list(i1=c(3,4,5),i2=c(6,7,8,9))); L
```

```
$f
[1] 1 2

$s
$s$i1
[1] 3 4 5

$s$i2
[1] 6 7 8 9
```

```
unlist(L)
```

```
 f1  f2 s.i11 s.i12 s.i13 s.i21 s.i22 s.i23 s.i24
  1   2     3     4     5     6     7     8     9
```

```
unlist(L,use.names=FALSE)
```

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

### 2.2.4 Фреймы

Фрейм данных — базовая структура R: список столбцов одинаковой длины с уникальными именами. Фрейм можно представить в виде прямоугольной таблицы с именованными столбцами. Если имена строк не заданы, то конкретная строка идентифицируется по её номеру.

Создаёт фрейм функция data.frame:

```
data.frame(..., row.names = NULL, check.rows = FALSE,
           check.names = TRUE, fix.empty.names = TRUE,
           stringsAsFactors = FALSE)
```

где

- `...` — аргумент вида `значение` или `тэг = значение`. Названия компонент фрейма создаются по тэгам (если они есть) или на основе анализа самого аргумента.
- `row.names` — NULL, целое число или строка символов указывающие столбец, содержимое которого будет использоваться в качестве имён строк. Если аргументом является вектор целых чисел или символов, то он сам будет использоваться для именования строк.
- `check.rows` — если TRUE, то строки проверяются на связность длин и названий.
- `check.names` — если TRUE, то названия переменных проверяются на синтаксическую правильность (можно ли их использовать в качестве имён переменных) и отсутствие дублей.
- `fix.empty.names` — логическое значение, определяющее, будут ли для неименованных столбцов конструироваться названия, или они останутся в виде пустых строк "".
- `stringsAsFactors` — логическое значение, определяющее, следует ли векторы символов рассматривать как качественные величины.

```
data.frame(x0 = 1,
           x1 = 1:10,
           x2 = sapply(1:10,function(x) x^2)) -> d; d
```

```
   x0 x1  x2
1   1  1   1
2   1  2   4
3   1  3   9
...
10  1 10 100
```

```
d[,2]              # второй столбец
d$x1               # столбец с именем x1
d[,c(1,3)]         # первый и третий столбцы

d[4:7,]            # строки с четвёртой по седьмую

d[4:7,c(1,3)]      # строки с четвёртой по седьмую из первого и третьего столбцов
```

### 2.2.5 Функции над структурами данных

Функция aggregate разбивает данные на подмножества, вычисляет по ним статистики и возвращает результат в подходящей форме:

```
aggregate(x, by, FUN, ..., simplify = TRUE, drop = TRUE)
```

где

- x — объект R.
- by — список элементов, по которым будет выполняться группировка. Список имеет столько же элементов, сколько их находится в x. Перед использованием группирующие элементы преобразуются в качественные величины.
- FUN — функция, применяемая к группе элементов.
- simplify — следует ли результат по возможности упрощать до вектора или матрицы?
- drop — следует ли отбрасывать необычные комбинации группирующих значений?

```
aggregate(data.frame(x=1:100),
          by=list(type=rep(c('odd','even'),50)),
          mean)
```

```
  type  x
1 even 51
2  odd 50
```

### Функции над векторами и списками

Функция `lapply` применяет к каждому элементу вектора или списка `x` функцию `fun` и возвращает результат в виде списка. `...` — необязательные аргументы для `fun`:

```
lapply(x, fun, ...)
```

```
x <- 1:10
square <- function(x) x^2
lapply(x, square)                    # список
unlist(lapply(x, square))            # вектор

L <- list(a=1, b=c(2,3), c=c(4,5,6))
lapply(L, mean)                      # список
```

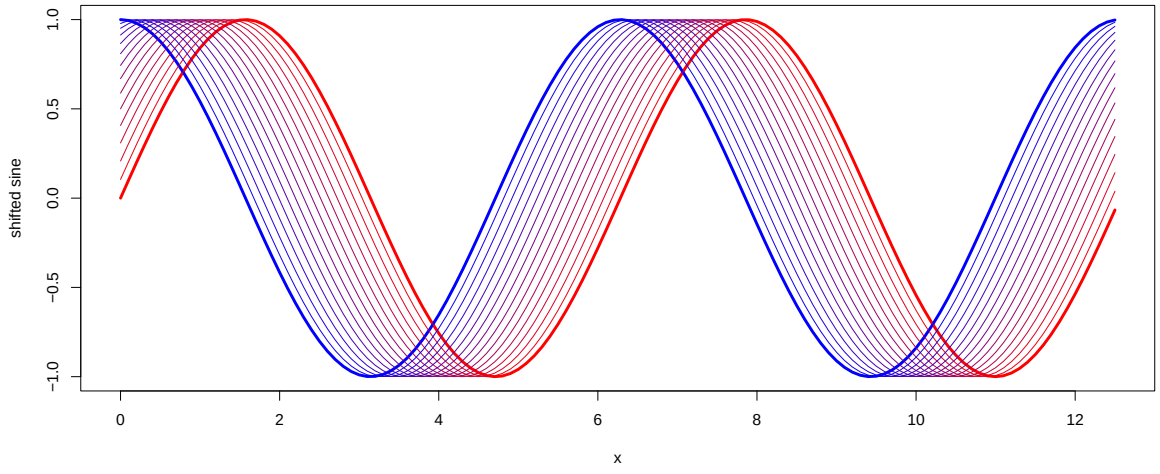
Функция `sapply` возвращает результат в виде вектора, матрицы или массива:

```
sapply(x, square)
sapply(L, mean)
```

**Пример.** *С помощью сдвига аргумента синус превращается в косинус:*

$$\sin(t + \pi/2) = \cos(t).$$

```
x <- seq(from=0,to=4*pi,by=0.1)        # 0 ⩽ x ⩽ 4π
fn <- function(t,shift) sin(t+shift)   # (t, φ) → sin(t + φ)

plot(x,sin(x),type='n',ylab='shifted␣sine')
lines(x,sin(x),col='red',lwd=3)   # из синуса
lines(x,cos(x),col='blue',lwd=3)  # в косинус

n <- 15 # n шагов
for (i in seq(from=0,to=1,by=1/n))
  lines(x,sapply(x, fn, pi/2*i),col=rgb(1-i,0,i))
```




❑

#### Функции над массивами

Массив может иметь несколько измерений, и параметр `margin` в функции `apply` указывает, по каким измерениям будет выполняться агрегирование:

```
apply(x, margin, fun, ..., simplify = TRUE)
```

где

- `x` — массив данных.
- `margin` — вектор, указывающий, какие измерения массива в качестве аргументов функции `fun`.

  Если `x` — матрица, то `1` — строки, `2` — стобцы, `c(1,2)` — строки и столбцы (вся матрица).
- `fun` — функция.
- `...` — необязательные аргументы функции `fun`.
- `simplify` — следует ли пытаться упростить результат?

```
x <- 1:10
x <- cbind(1,x,x^2,x^3)    # x_ij = i^(j-1)

apply(x,1,mean)            # x̄_i·
apply(x,2,mean)            # x̄_·j
apply(x,c(1,2),mean)       # x̄_ij
```

### 2.2.6 Другие типы данных

#### Дата и время

Для представления календарных дат используется класс `Date`. Для хранения даты и времени используются классы `POSIXct` и `POSIXlt`:

```
Sys.Date()                          # 2022-03-29
format(Sys.Date(), '%A,␣%B␣%d␣%Y')  # вторник, Март 29 2022
as.POSIXlt(Sys.Date())              # 2022-03-29 UTC
as.POSIXct(Sys.Date())              # 2022-03-29 05:00:00 +05
weekdays(Sys.Date())                # вторник
Sys.time()                          # 2022-03-29 14:53:20 +05
```

## 2.3 Случайные величины

Для работы со случайными величинами в `R` применяются функции, названия которых состоят из префикса и названия распределения.

#### Префиксы функций:

- `d` — функция плотности вероятности.
- `p` — функция распределения вероятностей.
- `q` — обратная функция распределения вероятностей.

- `r` — генератор значений случайной величины с заданным распределением.

Например, `dnorm` — функция плотности вероятности для нормального распределения; `runif` — генерация значений случайной величины, имеющей равномерное распределение.

### 2.3.1 Генерация случайных чисел

Состояние генератора случайных чисел (Random Number Generator, RNG) содержится в векторе `.Random.seed`. Это состояние может быть сохранено, а затем восстановлено, но пользователю не следует его *менять*.

Для настройки генератора можно воспользоваться функциями `RNGkind` и `RNGversion`:

```
RNGkind(kind = NULL, normal.kind = NULL, sample.kind = NULL)
RNGversion(vstr)
```

Функция `set.seed` используется при инициализации генератора:

```
set.seed(seed, kind = NULL, normal.kind = NULL, sample.kind = NULL)
```

Здесь

- `kind` — строка, определяющая желаемый тип генератора;
- `normal.kind` — тип генератора случайных чисел, имеющих нормальное распределение;
- `sample.kind` — тип генератора целых случайных чисел;
- `seed` — инициализирующее значение, число (если `seed = NULL`, то выполняется реинициализация генератора);
- `vstr` — строка, содержащая номер версии.

### 2.3.2 Дискретные распределения

#### Биномиальное распределение

Если в одном испытании вероятность успеха $p$ $(0 \leqslant p \leqslant 1)$, то вероятность $m$ успехов в серии из $n$ независимых испытаний $(0 \leqslant m \leqslant n)$ описывается выражением

$$C_n^m p^m (1-p)^{n-m},$$

где $C_n^m$ — биномиальный коэффициент (см. стр. 29).

Математическое ожидание и дисперсия:

$$\mathsf{E}\,\xi = np, \quad \mathsf{Var}\,\xi = np(1-p).$$

Свойства биномиального распределения (распределения Бернулли) подробно описаны в [10: с. 84–88].

```
dbinom(x, size, prob)
pbinom(q, size, prob)
qbinom(p, size, prob)
rbinom(n, size, prob)
```

- `size` — количество испытаний в серии.

- `prob` — вероятность успеха в одном испытании.

**Пример.** *Моделирование результатов бросков одной монеты и нескольких монет.*

Моделирование десяти бросков монеты:

```
rbinom(10,1,.5)
```

Моделирование результатов $n$ бросков $m$ монет: количество орлов (или решек) в серии:

```
n <- 10; m <- 5
rbinom(n,m,.5)
```

❑

#### Распределение Пуассона

Если длина временного интервала между случайными событиями — случайная величина, имеющая экспоненциальное распределение с параметром $\lambda$, то количество событий, которые произойдут в течение времени $t$, — случайная величина, имеющая распределение Пуассона с параметром $t\lambda$.

Вероятность того, что произойдёт $n$ событий за единицу времени:

$$\frac{\lambda^n \exp(-\lambda)}{n!}, \qquad n = 0, 1, 2, \ldots, \quad \lambda > 0.$$

Математическое ожидание и дисперсия:

$$\mathsf{E}\,\xi = \mathsf{Var}\,\xi = \lambda.$$

Свойства распределения Пуассона подробно описаны в [10: с. 88–90].

```
dpois(x, lambda)
ppois(q, lambda)
qpois(p, lambda)
rpois(n, lambda)
```

- `lambda` — вектор средних (неотрицательные значения).

### 2.3.3 Непрерывные распределения

#### Равномерное распределение

$\xi \sim \mathcal{U}(a, b)$ — случайная величина $\xi$ с равной вероятностью принимает любые значения из интервала $[a, b]$.

Плотность вероятности:

$$d_\xi(x) = \begin{cases} 1/(b-a), & a \leqslant x \leqslant b, \\ 0, & x < a \text{ или } x > b. \end{cases}$$

Математическое ожидание и дисперсия:

$$\mathsf{E}\,\xi = \frac{1}{2}(a+b), \quad \mathsf{Var}\,\xi = \frac{1}{12}(b-a)^2.$$

```
dunif(x, min = 0, max = 1)
punif(q, min = 0, max = 1)
qunif(p, min = 0, max = 1)
runif(n, min = 0, max = 1)
```

**Дискретное равномерное распределение.** Пусть требуется сгенерировать выборку для *дискретной* равномерно распределённой случайной величины $\eta$ со значениями из множества $\{m_1, m_2, \ldots, m_n\}$ ($m_i \neq m_j$ при $i \neq j$):

$$\mathsf{P}\{\eta = m_i\} = \frac{1}{n}.$$

Тогда

$$\eta = m_{\lfloor n\xi \rfloor + 1}, \quad \xi \in U(0,1).$$

Реализация генератора в виде функции:

```
rdunif <- function(n,min=0,max=9) trunc(min+runif(n)*(max+1-min))
```

и его использование

```
rdunif(100)                    # m = {0,1,...,9}
rdunif(100,min=1)              # m = {1,2,...,9}
rdunif(100,max=10)             # m = {0,1,...,10}
rdunif(100,min=1,max=10)       # m = {1,2,...,10}

m <- (1:10)^2                  # m_i = i^2, i = 1,2,...,10
m[rdunif(15,min=1,max=length(m))]
```

**Пример.** *Моделирование $n$ бросков игральной кости с $k$ гранями.*

Предполагаем, что игральная кость — правильный $k$-гранник: результат бросания $m_i = i$, $i = 1, \ldots, k$, и $\mathsf{P}\{m = i\} = 1/k$:

```
dice <- function(n=1, facets=6) rdunif(n,min=1,max=facets)
```

Серии из $n$ бросков:

```
n <- 10
dice(n,4)   # тетраэдр
dice(n)     # кубик
dice(n,37)  # правильный 37-гранник
```

❑

### Нормальное распределение

$\xi \sim \mathcal{N}(\mu, \sigma^2)$ — случайная величина $\xi$ имеет нормальное распределение (распределение Гаусса) с математическим ожиданием $\mu$ и дисперсией $\sigma^2$.

Согласно центральной предельной теореме [3: гл. 8], сумма достаточно большого количества слабо зависимых однородных (имеющих примерно одинаковый масштаб) случайных величин имеет распределение близкое к нормальному.

Плотность вероятности:

$$d_\xi(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right), \qquad x \in \mathbb{R}.$$

Математическое ожидание и дисперсия:

$$\mathsf{E}\,\xi = \mu, \quad \mathsf{Var}\,\xi = \sigma^2.$$

```
dnorm(x, mean = 0, sd = 1)
pnorm(q, mean = 0, sd = 1)
qnorm(p, mean = 0, sd = 1)
rnorm(n, mean = 0, sd = 1)
```

- `mean` — математическое ожидание $\mu$.
- `sd` — среднеквадратическое отклонение $\sigma$.

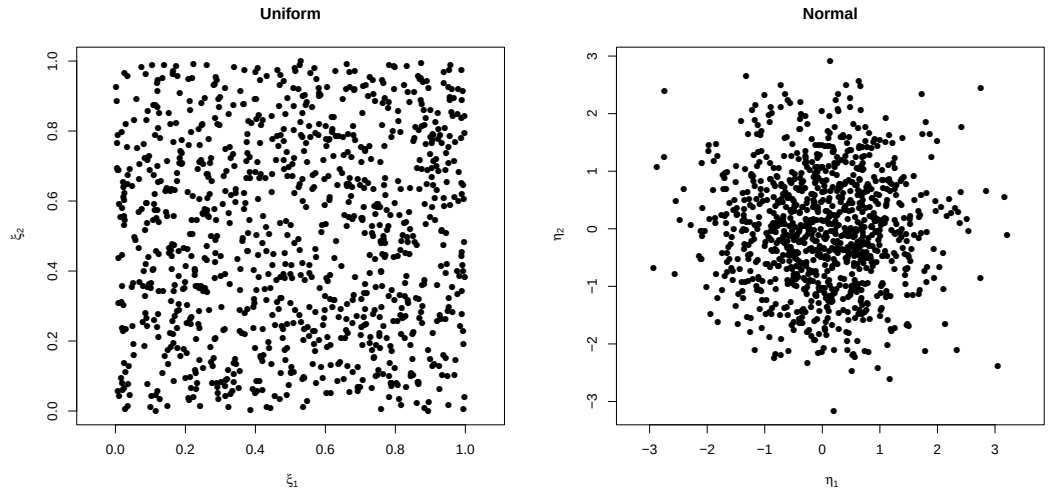
**Преобразование Бокса-Мюллера.** Пусть $\xi_1 \sim U(0,1)$ и $\xi_2 \sim U(0,1)$ — независимые случайные величины. Тогда

$$\eta_1 = \cos(2\pi\xi_1)\sqrt{-2\ln\xi_2} \quad \text{и} \quad \eta_2 = \sin(2\pi\xi_1)\sqrt{-2\ln\xi_2}$$

— независимые случайные величины, имеющие стандартное нормальное распределение ($\mu = 0$ и $\sigma^2 = 1$).

```
n <- 1000

x1 <- runif(n)
x2 <- runif(n)

plot(x1,x2,pch=16,asp=1,main='Uniform',
     xlab=expression(~xi[1]),ylab=expression(~xi[2]))

y1 <- cos(2*pi*x1)*(-2*log(x2))^0.5
y2 <- sin(2*pi*x1)*(-2*log(x2))^0.5

plot(y1,y2,pch=16,asp=1,main='Normal',
     xlab=expression(~eta[1]),ylab=expression(~eta[2]))
```

Облака точек, координаты которых являются случайными числами с равномерным (слева) и нормальным (справа) распределениями:




### Логарифмически нормальное (логнормальное) распределение

Если случайная величина имеет нормальное распределение $\eta \sim \mathcal{N}(\mu, \sigma^2)$, то её экспонента будет логарифмически нормальна: $\exp(\eta) = \xi \sim \mathcal{L}og\mathcal{N}(\mu, \sigma^2)$.

Очевидно, что если $\xi \sim \mathcal{L}og\mathcal{N}(\mu, \sigma^2)$, то $\ln\xi \sim \mathcal{N}(\mu, \sigma^2)$.

```
dlnorm(x, meanlog = 0, sdlog = 1)
plnorm(q, meanlog = 0, sdlog = 1)
qlnorm(p, meanlog = 0, sdlog = 1)
rlnorm(n, meanlog = 0, sdlog = 1)
```

- `meanlog` — $\mu$.
- `sdlog` — $\sigma$.

Плотность вероятности:

$$d_\xi(x) = \frac{1}{x\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(\ln x - \mu)^2}{2\sigma^2}\right), \qquad 0 < x.$$

Математическое ожидание и дисперсия:

$$\mathsf{E}\,\xi = \exp\left(\mu + \frac{\sigma^2}{2}\right), \quad \mathsf{Var}\,\xi = \left(\exp(\sigma^2) - 1\right)\exp(2\mu + \sigma^2).$$

**Пример** (линеаризуемое уравнение регрессии)**.**

Если при моделировании объём производства описывается производственной функцией Кобба-Дугласа

$$\beta_0 K^{\beta_1} L^{\beta_2},$$

где $K$ — объём основных фондов, а $L$ описывает трудовые ресурсы (затраты рабочего времени), то уравнение регрессии с *мультипликативной* ошибкой имеет вид

$$y = \beta_0 K^{\beta_1} L^{\beta_2} \varepsilon. \tag{2.4}$$

Если $\varepsilon \sim \mathcal{L}og\mathcal{N}(0, \sigma^2)$, то, логарифмированием обеих частей уравнения, (2.4) можно свести к форме множественной *линейной* регрессии:

$$\underbrace{\ln y}_{y'} = \ln\beta_0 + \beta_1 \underbrace{\ln K}_{K'} + \beta_2 \underbrace{\ln L}_{L'} + \underbrace{\ln \varepsilon}_{\varepsilon'},$$

где $\varepsilon' \sim \mathcal{N}(0, \sigma^2)$.

*Замечание.* Уравнение регрессии

$$y = \beta_0 K^{\beta_1} L^{\beta_2} + \varepsilon$$

с *аддитивной* ошибкой нелинеаризуемо. Если $\mathsf{E}\,\varepsilon = 0$, то для оценки значений коэффициентов модели можно использовать *нелинейный* метод наименьших квадратов.

*Замечание.* Для оценки коэффициентов в (2.4) можно использовать метод максимального правдоподобия.

❑

### Экспоненциальное распределение

Если количество событий, которые произойдут за единицу времени, — случайная величина, имеющая распределение Пуассона с параметром $1/\lambda$, то длина временного интервала между случайными событиями — случайная величина, имеющая экспоненциальное распределение с параметром $\lambda$.

$\xi \sim \mathcal{E}xp(\lambda)$ — случайная величина $\xi$ имеет экспоненциальное распределение с параметром $\lambda$ (величина, обратная значению математического ожидания: $\mathsf{E}\,\xi = 1/\lambda$).

```
dexp(x, rate = 1)
pexp(q, rate = 1)
qexp(p, rate = 1)
rexp(n, rate = 1)
```

- `rate` — $\lambda$.

Плотность вероятности:

$$d_\xi(x) = \begin{cases} \lambda \exp(-\lambda x), & 0 \leqslant x, \\ 0, & x < 0. \end{cases}$$

Математическое ожидание и дисперсия:

$$\mathsf{E}\,\xi = \frac{1}{\lambda}, \quad \mathsf{Var}\,\xi = \frac{1}{\lambda^2}.$$

Если $\eta \sim \mathcal{U}(0,1)$, то

$$-\frac{1}{\lambda}\ln\eta \sim \mathcal{E}xp(\lambda).$$

### $t$-распределение Стьюдента

Если $\xi_0, \xi_1, \ldots, \xi_n \sim \mathcal{N}(0,1)$ — независимые случайные величины, то

$$\xi = \frac{\xi_0}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}\xi_i^2}}$$

имеет $t$-распределение Стьюдента с $n$ степенями свободы: $\xi \sim t(n)$.

```
dt(x, df)
pt(q, df)
qt(p, df)
rt(n, df)
```

- `df` — число степеней свободы $n$.

Плотность вероятности ($\Gamma$ — гамма-функция, см. с. 29):

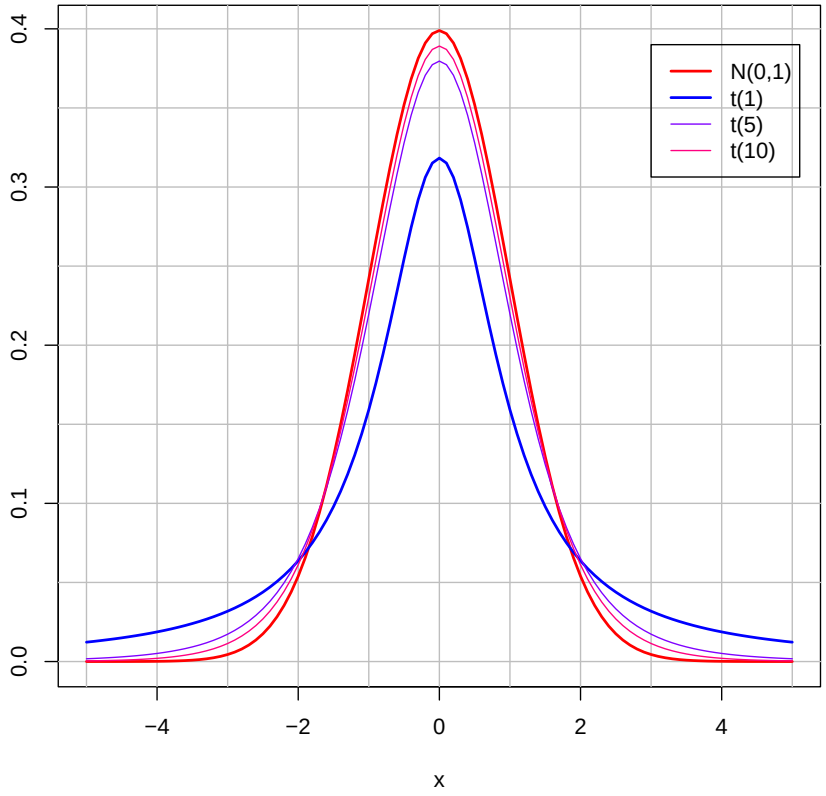
$$d_\xi(x) = \frac{\Gamma\left(\frac{n+1}{2}\right)}{\sqrt{n\pi}\,\Gamma\left(\frac{n}{2}\right)}\left(1+\frac{x^2}{n}\right)^{-\frac{n+1}{2}}, \qquad x \in \mathbb{R}.$$

Математическое ожидание и дисперсия:

$$\mathsf{E}\,\xi = 0, \quad \mathsf{Var}\,\xi = \frac{n}{n-2} \text{ при } n > 2.$$

**Пример.** *Сравнение графиков функций плотности вероятности для стандартного нормального распределения $\mathcal{N}(0,1)$ и $t$-распределений Стьюдента с 1, 5 и 10 степенями свободы.*

```
x <- seq(from=-5,to=5,by=0.1)
y <- matrix(nrow=4,ncol=length(x))

y[1,] <- dnorm(x)   # N(0,1)
y[2,] <- dt(x,1)    # t(1)
y[3,] <- dt(x,5)    # t(5)
y[4,] <- dt(x,10)   # t(10)

# холст
plot(c(min(x),max(x)),c(0,max(y)),xlab='x',ylab='',type='n')
# сетка
for (z in seq(from=0,to=0.4,by=0.05)) abline(h=z,col='grey')
for (z in seq(from=min(x),to=max(x),by=1)) abline(v=z,col='grey')
# толщина и цвет линий
LWD <- c(2,2,1,1)
COL <- c('red','blue',rgb(.5,0,1),rgb(1,0,.5))

for (i in 1:4) lines(x,y[i,],col=COL[i],lwd=LWD[i])

legend(3,0.39,
       legend=c('N(0,1)','t(1)','t(5)','t(10)'),
       col=COL,lty=1,lwd=LWD)
```




❑

### Распределение $\chi^2$

Если $\xi_1, \ldots, \xi_n \sim \mathcal{N}(0,1)$ — независимые случайные величины, то

$$\xi = \sum_{i=1}^{n} \xi_i^2$$

имеет распределение $\chi^2$ с $n$ степенями свободы: $\xi \sim \chi^2(n)$.

```
dchisq(x, df)
pchisq(q, df)
qchisq(p, df)
rchisq(n, df)
```

- `df` — число степеней свободы $n$.

Плотность вероятности ($\Gamma$ — гамма-функция, см. с. 29):

$$d_\xi(x) = \frac{(1/2)^{\frac{n}{2}}}{\Gamma\left(\frac{n}{2}\right)} x^{\frac{n}{2}-1} \exp\left(-\frac{x}{2}\right), \qquad x \in \mathbb{R}.$$

Математическое ожидание и дисперсия:

$$\mathsf{E}\,\xi = n, \quad \mathsf{Var}\,\xi = 2n.$$

### $F$-распределение Фишера

Если $\xi_1 \sim \chi^2(n_1)$ и $\xi_2 \sim \chi^2(n_2)$ — независимые случайные величины, то

$$\xi = \frac{\xi_1/n_1}{\xi_2/n_2}$$

имеет распределение Фишера с $n_1$ и $n_2$ степенями свободы: $\xi \sim \mathcal{F}(n_1, n_2)$.

```
df(x, df1, df2)
pf(q, df1, df2)
qf(p, df1, df2)
rf(n, df1, df2)
```

- `df1` и `df2` — число степеней свободы $n_1$ и $n_2$.

Математическое ожидание и дисперсия:

$$\mathsf{E}\,\xi = \frac{n_2}{n_2 - 2} \text{ при } n_2 > 2, \quad \mathsf{Var}\,\xi = \frac{2n_2^2(n_1 + n_2 - 2)}{n_1(n_2 - 2^2)(n_2 - 4)} \text{ при } n_2 > 4.$$

## 2.3.4 Статистики

### Ожидаемая величина

Если $\xi$ — случайная величина, то чего от неё можно ожидать? На значение какой *неслучайной* величины следует ориентироваться в ходе рассуждений?

**Математическое ожидание** Чаще всего в роли ожидаемой величины используется математическое ожидание, значение которого определяется выражениями

$$\mathsf{E}\,\xi = \int_{-\infty}^{\infty} x d_\xi(x)\,dx \quad \text{или} \quad \mathsf{E}\,\xi = \sum_i x_i\,\mathsf{P}\{\xi = x_i\}$$

в зависимости от того, является $\xi$ непрерывной или дискретной случайной величиной.

Выборочной оценкой значения математического ожидания служит среднее значение.

*Замечание.* Средние значения можно использовать только для количественных величин! Вычисление среднего арифметического для порядковых (ранговых и тем более номинальных) величин — **грубейшая ошибка**!!!

### Среднее арифметическое

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i.$$

```
sum(x)/length(x)
mean(x)                    # встроенная функция
```

Если $X$ — матрица, то функции `rowMeans` и `colMeans` позволяют вычислить средние арифметические по столбцам и строкам соответственно:

```
x <- 1:10
X <- cbind(1,x,x^2)        # X_ij = i^(j-1)

rowMeans(X)                # X̄_i.
colMeans(X)                # X̄_.j
```

Аналогичный результат можно получить с помощью функции `apply`:

```
apply(X,1,mean)            # X̄_i.
apply(X,2,mean)            # X̄_.j
```

Взвешенное среднее арифметическое возвращается функцией `weighted.mean`:

$$\bar{x}_w = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i}.$$

```
x <- 1:10
w <- -(-10:-1)
sum(w*x)/sum(w)
weighted.mean(x,w)         # встроенная функция
```

### Среднее геометрическое

$$\bar{x}_g = \sqrt[n]{\prod_{i=1}^{n} x_i} = \exp\left(\frac{1}{n}\sum_{i=1}^{n} \ln x_i\right).$$

```
prod(x)^(1/length(x))
exp(sum(log(x))/length(x))     # дольше, но надёжнее
```

**Среднее гармоническое**

$$\bar{x}_h = \frac{n}{\sum_{i=1}^{n}(1/x_i)}.$$

```
length(x)/sum(1/x)
```

**Медиана** Пусть $x_1, x_2, \ldots, x_n$ — числа, отсортированные по возрастанию (или по убыванию). Тогда

$$\text{медиана}(x) = \begin{cases} x_{\frac{n+1}{2}}, & n \text{ — нечётное}, \\ (x_{\frac{n}{2}} + x_{\frac{n}{2}+1})/2, & n \text{ — чётное}. \end{cases}$$

```
odd <- function(n) n%%2==1        # x — нечётное?

n <- 11
x <- sort(runif(n)); x
if (odd(n)) x[(n+1)%/%2] else (x[n%/%2]+x[n%/%2+1])/2
```

Имеется также встроенная функция `median`:

```
median(x)
```

*Замечание.* Значение медианы устойчиво к добавлению в выборку небольшого числа очень больших или очень маленьких значений. Медиану следует использовать, если элементы выборки имеют существенно различающиеся по величине значения.

**Мода** — наиболее часто встречающаяся в выборке величина. В `R` нет встроенной функции для вычисления моды, поэтому создадим её самостоятельно:

```
modalValue <- function(x)
{
  # выделяем уникальные значения...
  u <- unique(x)
  # ...вычисляем их частоты...
  freq <- sapply(u,function(u,x) sum(x==u),x)
  # ...и определяем значения с максимальной частотой
  u[which(freq==max(freq))]
}
```

Здесь

- для выборки уникальных значений применяется функция `unique`;
- вычисление частот уникальных значений выполняется функцией `sapply`;
- местоположение величин, имеющих наибольшую частоту, определяется функцией `which`, которая может при необходимости возвращать несколько индексов.

  Функция `which.max` при наличии нескольких индексов возвращает только первый из них.

Пример использования (функция `rdunif` определена на с. 43):

```
modalValue(rdunif(100,1,10))
```

### Разброс

Оценка отклонения случайной величины относительно среднего значения или расстояния от минимального значения до максимального. Может использоваться для оценки неопределённости значения используемой величины.

**Дисперсия** — математическое ожидание квадрата отклонения значения случайной величины от математического ожидания этой величины:

$$\mathsf{Var}\,\xi = \mathsf{E}(\xi - \mathsf{E}\,\xi)^2.$$

Среднеквадратическое (или стандартное) отклонение — квадратный корень из дисперсии:

$$\sigma_\xi = \sqrt{\mathsf{Var}\,\xi}.$$

**Выборочная дисперсия** и выборочное среднеквадратическое (стандартное) отклонение $x$:

$$s_x^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2, \quad s_x = \sqrt{s_x^2}.$$

Для вычисления выборочных дисперсий и стандартного отклонения используются функции `var` и `sd` соответственно. Конечно, можно использовать и явные выражения (в частности, если следует уменьшать количество степеней свободы не на единицу, а на другое число):

```
sum((x-mean(x))^2)/(length(x)-1)      # s_x^2
var(x)                                # встроенная функция

sum((x-mean(x))^2)/(length(x)-1)^.5   # s_x
sd(x)                                 # встроенная функция
```

*Замечание.* Функции `var` и `sd` предполагают, что количество степеней свободы должно быть уменьшено на единицу. Иногда это предположение не выполняется, например, при вычислении дисперсии ошибки модели (множественной) линейной регрессии. В этом случае можно использовать явное выражение или исправить результат встроенной функции.

**Размах** — разница между наибольшим и наименьшим значениями в выборке:

$$\max_i x_i - \min_i x_i.$$

```
max(x)-min(x)
```

Вместо вычисления максимального (`max`) и минимального (`min`) значений в выборке по отдельности, можно использовать функцию `range`, возвращающую вектор, первым элементом которого является нижняя граница значений, а вторым — верхняя.

```
range(x)
```

Функция `diff`, используемая для вычисления разностей, позволяет найти размах:

```
diff(range(x))
```

### Коэффициент эксцесса

Иначе, коэффициент островершинности — мера остроты пика распределения случайной величины относительно нормального распределения:

$$\gamma_\xi = \frac{1}{\sigma_\xi^4}\mathsf{E}(\xi - \mathsf{E}\,\xi)^4 - 3.$$

Если $\gamma_\xi > 0$, то пик распределения вблизи математического ожидания более острый, чем у нормального распределения; если $\gamma_\xi < 0$, то — более гладкий.

```
x <- rnorm(1000)

sum((x-mean(x))^4)/(length(x)-1) / sd(x)^4 - 3
```

В библиотеке `moments` имеется функция `kurtosis`, которая оценивает эксцесс по формуле

$$\frac{\frac{1}{n}\sum_{i=1}^n (x_i - \bar{x})^4}{\left(\frac{1}{n}\sum_{i=1}^n (x_i - \bar{x})^2\right)^2}.$$

```
library(moments)
kurtosis(x)
# аналогично
mean((x-mean(x))^4) / mean((x-mean(x))^2)^2
```

### Коэффициент асимметрии

Несимметричность распределения значений случайной величины относительно математического ожидания:

$$\frac{1}{\sigma_\xi^3}\mathsf{E}(\xi - \mathsf{E}\,\xi)^3.$$

```
x <- rnorm(1000)

sum((x-mean(x))^3)/(length(x)-1) / sd(x)^3
```

В библиотеке `moments` имеется функция `skewness`, которая оценивает асимметрию по формуле

$$\frac{\frac{1}{n}\sum_{i=1}^n (x_i - \bar{x})^3}{\left(\frac{1}{n}\sum_{i=1}^n (x_i - \bar{x})^2\right)^{3/2}}.$$

```
library(moments)
skewness(x)
# аналогично
mean((x-mean(x))^3) / mean((x-mean(x))^2)^(3/2)
```

### Связь между величинами $\xi$ и $\eta$

**Ковариация** описывает тенденцию изменения $\eta$ при изменении $\xi$:

$$\mathrm{Cov}(\xi, \eta) = \mathsf{E}(\xi - \mathsf{E}\,\xi)(\eta - \mathsf{E}\,\eta).$$

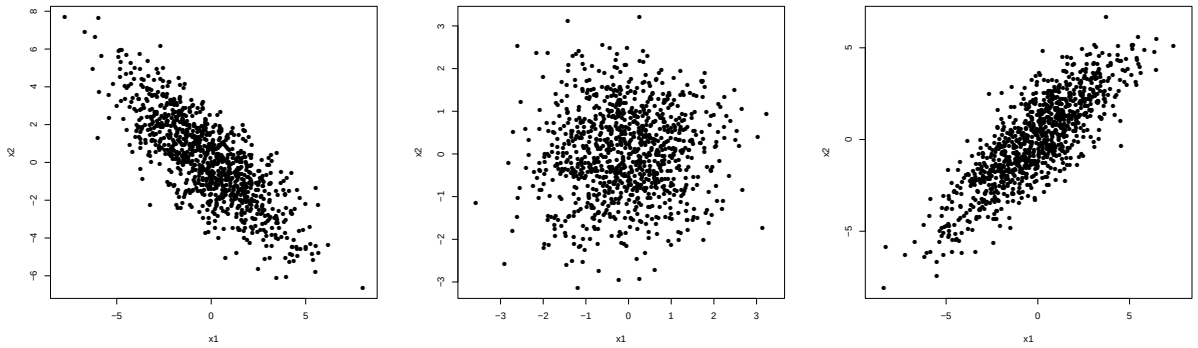
Для вычисления выборочной ковариации

$$\mathrm{cov}(x, y) = \frac{1}{n-1}\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$$

можно использовать функцию `cov`:

```
n <- 1000
x <- rnorm(n)
y <- x+rnorm(n)

cov(x,y)
# аналогично
sum((x-mean(x))*(y-mean(y)))/(n-1)
```

Графики рассеяния при Cov < 0 (отрицательная связь, слева), Cov = 0 (нет связи) и Cov > 0 (положительная связь, справа):




**Парная линейная корреляция** безразмерное значение в интервале $[-1, 1]$, описывающее силу тенденции изменения $\eta$ при изменении $\xi$:

$$\rho(\xi, \eta) = \frac{\mathrm{Cov}(\xi, \eta)}{\sigma_\xi \, \sigma_\eta}$$

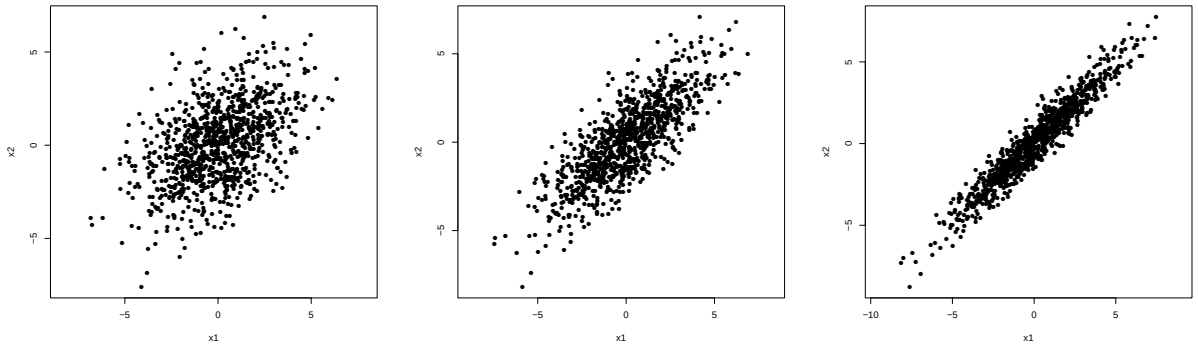
Для вычисления выборочной корреляции

$$r_{x,y} = \frac{\mathrm{cov}(x, y)}{s_x \, s_y}$$

можно использовать функцию `cor`:

```
cor(x,y)
# аналогично
sum((x-mean(x))*(y-mean(y)))/(sum((x-mean(x))^2)*sum((y-mean(y))^2))^.5
```

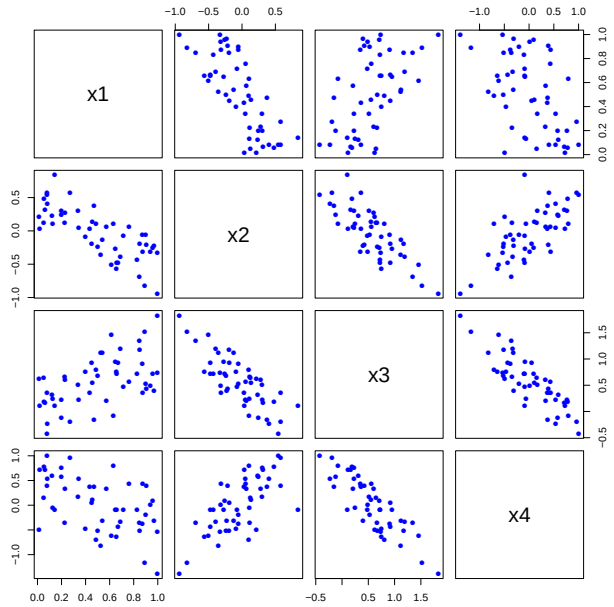
Графики рассеяния при $\rho \approx 0.5$, $\rho \approx 0.8$ и $\rho \approx 0.95$:




**Пример.** *Использование матричной арифметики для вычисления коэффициентов парной корреляции для выборок случайных величин* $\xi_1, \ldots, \xi_m$.

Пусть $x_1, x_2, x_3, x_4$ — выборки объёма $n$ для случайных величин $\xi_1, \xi_2, \xi_3, \xi_m$.

```
n <- 50
x1 <- runif(n)            # генератор ξ
x2 <- -x1 + runif(n)
x3 <- -x2 + runif(n)
x4 <- -x3 + runif(n)
plot(data.frame(x1=x1,x2=x2,x3=x3,x4=x4),pch=16,col='blue')
```




Объединим $x_1$, $x_2$, $x_3$ и $x_4$ по столбцам в матрицу $X$:

```
X <- cbind(x1,x2,x3,x4)
```

Сейчас $X_{ij}$ — $i$-й элемент $j$-й выборки. Нормируем и центрируем столбцы $X$, сохранив результат в матрице $Z$:

$$Z_{ij} = \frac{X_{ij} - \bar{x}_j}{s_{x_j}}.$$

Здесь $\bar{x}_j$ — среднее арифметическое, а $s_{x_j}$ — среднеквадратическое отклонение выборки $x_j$.

```
Z <- t(apply(X,1,function(x,m,s) (x-m)/s,
             apply(X,2,mean),apply(X,2,sd)))
```

Матрица коэффициентов парной линейной корреляции $R$ определяется выражением

$$R = \frac{1}{n-1} Z^\top Z.$$

```
R <- t(Z)%*%Z/(n-1); R
```

```
           x1         x2         x3         x4
x1  1.0000000 -0.7610361  0.5472509 -0.5173836
x2 -0.7610361  1.0000000 -0.7939531  0.7034447
x3  0.5472509 -0.7939531  1.0000000 -0.8474541
x4 -0.5173836  0.7034447 -0.8474541  1.0000000
```

Для проверки:

```
cor(x2,x4)        # 0.7034447
```

❑

**Частная линейная корреляция** полезна, если имеется несколько случайных величин $\xi_1, \xi_2, \ldots, \xi_k$ и требуется оценить корреляцию между, например, $\xi_1$ и $\xi_2$. Если просто вычислить $\rho(\xi_1, \xi_2)$, то на это значение могут оказать влияние остальные величины: $\rho(\xi_1, \xi_2 \mid \xi_3, \ldots, \xi_k)$, которые могут быть связаны как с $\xi_1$, так и с $\xi_2$.

Для очистки оценки зависимости между $\xi_i$ и $\xi_j$ от посторонних влияний воспользуемся выражением (см. [14: с. 215–216])

$$r_{ij} = \frac{A_{ij}}{\sqrt{A_{ii} A_{jj}}}.$$

Здесь $r_{ij}$ — оценка коэффициента частной корреляции между $\xi_i$ и $\xi_j$; $A_{ij}$, $A_{ii}$ и $A_{jj}$ — алгебраические дополнения к соответствующим элементам матрицы парных корреляций $R$.

**Пример** (Продолжение). *Вычисление матрицы частных корреляций.*

```
r <- array(dim=dim(R))
for (i in 1:dim(R)[1])
for (j in 1:dim(R)[2])
  r[i,j] <- det(R[-i,-j])/(det(R[-i,-i])*det(R[-j,-j]))^0.5
r
```

```
            [,1]        [,2]       [,3]        [,4]
[1,]  1.00000000 -0.63778968  0.1593716 -0.07834144
[2,] -0.63778968  1.00000000 -0.5002849 -0.02284652
[3,]  0.15937156 -0.50028494  1.0000000 -0.67059287
[4,] -0.07834144 -0.02284652 -0.6705929  1.00000000
```

❑

**Коэффициент множественной корреляции** $R_i^0$ показывает силу линейной связи случайной величины $\xi_i$ одновременно со всеми другими случайными величинами $\xi_j$, $j = 1, \ldots, k$, $j \neq i$ (см. [14: с. 216]):

$$R_j^0 = \sqrt{1 - \frac{\det R}{\det R_{jj}}}.$$

Здесь $\det R_{jj}$ — минор к матрице $R$.

**Пример** (Продолжение). *Оценка значений коэффициентов множественной корреляции.*

```
sapply(1:4,function(j,R) (1-det(R)/det(R[-j,-j]))^0.5,R)
```

```
[1] 0.7684316 0.8846995 0.8948917 0.8499569
```

❑

## 2.4 Чтение и запись

### 2.4.1 Соединение

Функция `file` позволяет установить соединение (по умолчанию не открытое) с файлом:

```
file(description = "", open = "", blocking = TRUE,
     encoding = getOption("encoding"), raw = FALSE,
     method = getOption("url.method", "default"))
```

Здесь

- `description` — путь к файлу.
- `open` — тип (модель) соединения:
  - `r` или `rt` — чтение текста;
  - `w` или `wt` — запись текста;
  - `a` или `at` — добавление текста в конец;
  - `rb` — чтение двоичных данных;
  - `wb` — запись двоичных данных;
  - `ab` — добавление двоичных данных;
  - `r+` или `r+b` — чтение и запись;
  - `w+` или `w+b` — чтение и запись с предварительной очисткой файла;
  - `a+` или `a+b` — чтение и добавление.
- `blocking` — в режиме блокировки управление не возвращается в `R` до завершения операции чтения, записи или добавления; в неблокирующем режиме управление возвращается максимально быстро (возможно, даже до завершения операции);
- `encoding` — кодировка текста, аналогично команде `iconv`.
- `raw` — использование интерфейса, более подходящего для нерегулярных (например, сжатых) файлов.

Функция `file` возвращает объект класса `connection`, который будет использоваться при открытии (функция `open`) и закрытии (функция `close`) соединения:

```
open(con, open = "r", blocking = TRUE, ...)
close(con, type = "rw", ...)
```

Функция `flush` выталкивает данные из потока в файл, открытый для записи или добавления:

```
flush(con)
```

Функция `isOpen` позволяет проверить состояние соединения:

```
isOpen(con, rw = "")
```

С помощью функции `url` можно устанавить соединение с файлом, используя полные URL (включая такие схемы, как `http://` или `https://`, `ftp://` или `file://`):

```
url(description, open = "", blocking = TRUE,
    encoding = getOption("encoding"),
    method = getOption("url.method", "default"),
    headers = NULL)
```

Здесь

- `description` — символьная строка, описывающая подключение. Для файлов — путь к файлу, который будет открыт, или полный URL.
- `open` — `"r"`, `"w"`, `"a"` и т.п.

Для работы с архивами применяются функции `gzfile` (архивы `gzip`), `bzfile` (архивы `bzip2`), `xzfile` (архивы `xz` или, только для чтения, `lzma`), а также `unz` (чтение архивов `zip`):

```
gzfile(description, open = "", encoding = getOption("encoding"), compression = 6)
bzfile(description, open = "", encoding = getOption("encoding"), compression = 9)
xzfile(description, open = "", encoding = getOption("encoding"), compression = 6)
unz(description, filename, open = "", encoding = getOption("encoding"))
```

### 2.4.2 Работа с данными

#### Чтение

Для чтения двоичных данных из файла используется функция `load`:

```
load(file, envir = parent.frame(), verbose = FALSE)
```

Функция `download.file` применяется для загрузки файла из интернета:

```
download.file(url, destfile, method, quiet = FALSE, mode = "w",
              cacheOK = TRUE,
              extra = getOption("download.file.extra"),
              headers = NULL, ...)
```

Здесь

- `method` — по умолчанию `"auto"` (соответствует `"internal"` для URL вида `file://` и `"libcurl"` для остальных URL).
  - `"libcurl"` поддерживает одновременные загрузки (векторы `url` и `destfile` могут иметь длину больше единицы).

    При одном URL и `quiet=FALSE` в интерактивном режиме показывается строка прогресса.
  - `"interval"` поддерживает только `file://`
  - `"wininet"` поддерживает `file://` и (с предупреждением) `http://` и `https://`.
  - `"wget"` и `"curl"` требуют, чтобы соответствующие программы[7][8] были установлены и могли быть найдены (для операционной системы Windows — путь к программе должет содержаться в переменной среды `PATH`).

    Блокируют всю активность `R`, что может сделать графический интерфейс программы недоступным на время работы команды.

    Параметр `extra` позволяет задать дополнительные аргументы командной строки для `wget` и `curl`.

Функции `read.table` (функция общего назначения), `read.csv` и `read.csv2` (comma-separated values), `read.delim` и `read.delim2` (данные с разделителями) применяются для чтения из файла данных, оформленных в виде таблицы, и создания фрейма данных:

```
read.table(file, header = FALSE, sep = "", quote = "\"'",
           dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),
           row.names, col.names, as.is = !stringsAsFactors,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
           allowEscapes = FALSE, flush = FALSE,
           stringsAsFactors = FALSE,
           fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)
```

Здесь

[7] https://gnuwin32.sourceforge.net/packages/wget.htm
[8] https://curl.se/windows/

- `file` — путь к файлу, из которого выполняется чтение данных. Также может быть объектом типа `connection`, созданным функцией `file`.
- `header` — содержит ли первая строка файла названия столбцов. Если `FALSE`, то предполагается, что в первой строке находятся названия столбцов тогда и только тогда, когда количество элементов в первой строке на единицу меньше числа столбцов.
- `sep` — символ-разделитель элементов в строке.
- `quote` — множество символов, которые могут быть использованы в качестве ограничителей при использовании строк.
- `dec` — символ-разделитель целой и дробной частей числа.

Например, для загрузки данных из файла `data.txt` можно использовать следующую последовательность действий:

```
f <- file('d:/data.txt',open='rt')
open(f)
read.table(f) -> dat
close(f)
```

или сделать всё одной командой:

```
read.table('d:/data.txt') -> dat
```

Чтение csv-файлов (Comma Separated Values, файл с разделителями-запятыми) и файлов с другими разделителями. В принципе, достаточно одного `read.csv` с явным заданием при необходимости разделителей элементов в строке и разделителей целой и дробной части чисел:

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

read.csv2(file, header = TRUE, sep = ";", quote = "\"",
          dec = ",", fill = TRUE, comment.char = "", ...)

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

read.delim2(file, header = TRUE, sep = "\t", quote = "\"",
            dec = ",", fill = TRUE, comment.char = "", ...)
```

Например, чтение из `csv`-файла с явным указанием символа-разделителя элементов в строке и символа-разделителя целой и дробной частей выполняется так:

```
read.csv('d:/wages.csv',sep=';',dec=',') -> dat
```

Далее можно указать, что в столбцах `SEX` и `EDU` находятся качественные данные:

```
# количественные → качественные
dat$SEX <- as.factor(dat$SEX)
dat$EDU <- as.factor(dat$EDU)
```

а затем ввести соответствующие метки для значений качественных величин:

```
levels(dat$SEX) <- c('male','female')
levels(dat$EDU) <- c('primary','high','techical','bachelor','postgrad')
```

Общая информация о загруженных и преобразованных данных:

```
summary(dat)
```

```
       W             SEX           EDU          AGE
 Min.   : 5.92   male  :75   primary :24   Min.   :17.00
 1st Qu.:15.95   female:75   high    :36   1st Qu.:25.00
 Median :20.30               techical:53   Median :31.00
 Mean   :21.78               bachelor:25   Mean   :33.11
 3rd Qu.:25.21               postgrad:12   3rd Qu.:39.75
 Max.   :81.67                             Max.   :61.00
```

### Запись

Функция `save` записывает в файл представление объекта `R`, которое в дальнейшем может быть загружено в `R` с помощью функции `load`:

```
save(..., list = character(),
     file = stop("'file' must be specified"),
     ascii = FALSE, version = NULL, envir = parent.frame(),
     compress = isTRUE(!ascii), compression_level,
     eval.promises = TRUE, precheck = TRUE)
```

**Пример.** *Запись в файл значений треугольника Паскаля* *(см. с. 36)*.

```
# треугольник Паскаля
n <- 10
pas <- list(c(1,1))
for (i in 1:(n-1))
  pas[[i+1]] <- c(pas[[i]][1],
                  pas[[i]][-1]+pas[[i]][-(i+1)],
                  pas[[i]][i+1])

# запись в двойчный файл
f <- file('d:/pascal.dat',open='wb')
save(pas,file=f)
close(f)

# запись в текстовый файл
f <- file('d:/pascal.txt',open='wt')
for (i in 1:n)
{
  cat('level:␣␣',i,'\n',file=f)
  cat('coeffs:␣',pas[[i]],'\n',file=f)
}
close(f)
```

Содержимое текстового файла `pascal.txt`:

```
level:   1
coeffs:  1 1
level:   2
coeffs:  1 2 1
level:   3
```

```
coeffs:  1 3 3 1
.............................
level:   10
coeffs:  1 10 45 120 210 252 210 120 45 10 1
```

❑

Функция save.image записывает в файл всё текущее окружение:

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

Для записи в файл данных в виде таблицы применяются функция write.table (наиболее общая), а также функции write.csv или write.csv2:

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

write.csv(...)
write.csv2(...)
```

### 2.4.3 Сценарии

Функция source заставляет R загрузить из внешнего источника данные, распознать их, а затем выполнить получившийся набор команд:

```
source(file, local = FALSE, echo = verbose, print.eval = echo,
       exprs, spaced = use_file,
       verbose = getOption("verbose"),
       prompt.echo = getOption("prompt"),
       max.deparse.length = 150, width.cutoff = 60L,
       deparseCtrl = "showAttributes",
       chdir = FALSE,
       encoding = getOption("encoding"),
       continue.echo = getOption("continue"),
       skip.echo = 0, keep.source = getOption("keep.source"))
```

## 2.5 Простейшие задачи

Решение уравнений и математическое программирование.

### 2.5.1 Решение нелинейных уравнений

#### Использование встроенных функций

Для решения систем нелинейных уравнений можно использовать функцию nleqslv из пакета nleqslv:

```
nleqslv(x, fn, jac=NULL, ...,
   method = c("Broyden", "Newton"),
   global = c("dbldog", "pwldog", "cline", "qline", "gline", "hook", "none"),
```

```
    xscalm = c("fixed","auto"),
    jacobian=FALSE,
    control = list())
```

Здесь

- `x` — начальная пробная точка.
- `fn` — вектор-функция, корень которой будет искаться.
- `jac` — функция, возвращающая значение матрицы Якоби в текущей точке.
- `method` — метод, используемый при решении.

  В методе Бройдена матрица Якоби, вычисленная на первом шаге, корректируется при следующих итерациях.

  В методе Ньютона матрица Якоби вычисляется на каждой итерации.
- `global` — метод, используемый для определения следующей пробной точки.
- `xscalm` — метод масштабирования `x`.
- `jacobian` — возвращать ли матрицу Якоби в результате.
- `control` — список управляющих параметров.

**Пример.** *Решение задачи условной минимизации*

$$f(x) = x_1^2 + 2x_2^2 \to \min,$$
$$h(x) = 3x_1 + x_2 - 5 = 0.$$

*методом множителей Лагранжа сводится к решению системы уравнений*

$$\frac{\partial F(x,\lambda)}{\partial x_1} = 0, \quad \frac{\partial F(x,\lambda)}{\partial x_2} = 0, \quad \frac{\partial F(x,\lambda)}{\partial \lambda} = 0,$$

*где* $F(x,\lambda)$ *— функция Лагранжа*

$$F(x,\lambda) = f(x) + \lambda h(x).$$

```
# Функция Лагранжа...
f <- function(arg)
{
  x <- arg[-length(arg)]
  lambda <- arg[length(arg)]

  sum(x^2*c(1,2))+lambda*(sum(x*c(3,1))-5)
}
# ...и оценка значения вектор-функции её частных производных
df <- function(x,h=0.01)
{
  res <- double(length(x))
  for (i in 1:length(x))
  {
    x[i] <- x[i] + h
    res[i] <- F(x)
```

```
    x[i] <- x[i] - h
    res[i] <- res[i] - F(x)
  }
  res/(2*h)
}

# Решение системы нелинейных уравнений
nleqslv(c(0,0,0),df)

$x
[1]  1.5802632  0.2592105 -1.0568421
$fvec
[1] -2.220446e-14  0.000000e+00  0.000000e+00
...
```

Использование явной вектор-функции частных производных:

```
df <- function(x) c(2*x[1]+3*x[3], 4*x[2]+x[3], 3*x[1]+x[2]-5)

nleqslv(c(0,0,0),df)

$x
[1]  1.5789474  0.2631579 -1.0526316
$fvec
[1] -2.220446e-15  4.440892e-16  8.881784e-16
...
```

❑

Другим вариантом поиска корней является сведéние решения уравнения или системы уравнений

$$f_i(x) = 0, \quad i = 1, \ldots, n,$$

к задаче минимизации

$$\sum_{i=1}^{n} f_i(x)^2 \to \min$$

и использование функции `optim`.

### Решение скалярного уравнения методом Ньютона

Решение нелинейного уравнения

$$f(x) = 0$$

с гладкой функцией $f\colon \mathbb{R} \to \mathbb{R}$ методом Ньютона заключается в выборе начальной пробной точки $x_0$ с последующим вычислением приближений $x_1, x_2, \ldots$ по формуле

$$x_i = x_{i-1} - \frac{f(x_{i-1})}{f'(x_{i-1})}, \quad i = 1, 2, \ldots \tag{2.5}$$

Для оценки значения производной будем использовать двухстороннюю аппроксимацию:

$$f'(x) \approx \frac{1}{2h}\big(f(x+h) - f(x-h)\big),$$

а также программирование в функциональном стиле, когда и на вход функции `D` поступает функция, и результат на выходе — функция:

```
# f'(x) ≈ (f(x+h) - f(x-h))/(2h)
D <- function(f,h=0.1) function(x) (f(x+h)-f(x-h))/(2*h)
```

Собственно решатель применяет коррекцию (2.5) пока оценка точности не достигнет заданного значения eps:

```
solver <- function(x0,f,eps=0.01)
{
  df <- D(f)
  newton <- function(x) x-f(x)/df(x)

  x <- x0
  repeat
  {
    x_next <- newton(x)

    if (abs(x-x_next)<eps && abs(f(x_next))<eps) return (x_next)
    x <- x_next
  }
}
```

**Пример.** *Решение уравнения* $\sin(x) = 0.5$.

```
f <- function(x) sin(x)-0.5
```

Поиск решения от начальной пробной точки $x_0 = 1$:

```
solver(1,f)
```

❏

**Пример.** *Оценка значений* всех *корней многочлена* $x^3 - 2 = 0$.

```
f <- function(x) x^3-2
```

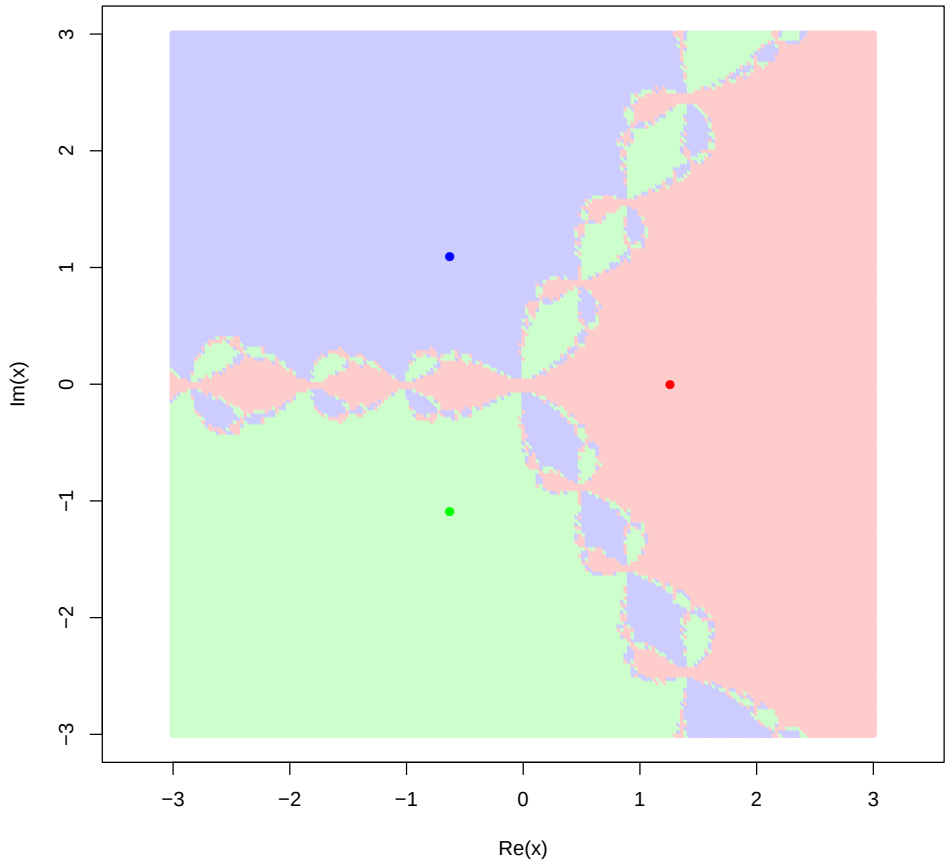
Уравнение имеет один вещественный корень и два комплексно-сопряжённых:

```
solver(1,f)       # 1.259942
solver(-1+1i,f)   # -0.62998 + 1.09108i
solver(-1-1i,f)   # -0.62998 - 1.09108i
```

*Замечание.* Вычисленное решение уравнения

$$x^2 = 2$$

существенно зависит от выбора начального приближения $x_0$, а области притяжения разделены фрактальной границей:




❑

### Решение системы уравнений методом Ньютона

Решение системы нелинейных уравнений

$$f(x) = 0$$

с гладкой вектор-функцией $f\colon \mathbb{R}^m \to \mathbb{R}^m$ методом Ньютона заключается в выборе начальной пробной точки $x_0 \in \mathbb{R}^m$ с последующим вычислением приближений $x_1, x_2, \ldots$ по формуле

$$x_0, \quad x_i = x_{i-1} + p_{i-1}, \quad i = 1, 2, \ldots,$$

где величина коррекции $p_{i-1}$ определяется при решении системы линейных алгебраических уравнений

$$\nabla f(x_{i-1}) p_{i-1} = -f(x_{i-1}).$$

Матрица первых производных (матрица Якоби) вектор-функции векторного аргумента $f\colon \mathbb{R}^n \to \mathbb{R}^m$ имеет вид

$$\nabla f = \begin{pmatrix} \frac{\partial f_1(x)}{\partial x_1} & \frac{\partial f_1(x)}{\partial x_2} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \frac{\partial f_2(x)}{\partial x_1} & \frac{\partial f_2(x)}{\partial x_2} & \cdots & \frac{\partial f_2(x)}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m(x)}{\partial x_1} & \frac{\partial f_m(x)}{\partial x_2} & \cdots & \frac{\partial f_m(x)}{\partial x_n} \end{pmatrix}.$$

Для оценки значения $\partial f_i / \partial x_j$ в точке $x$ можно использовать одностороннюю (погрешность $O(h)$) или двухстороннюю (погрешность $O(h^2)$) аппроксимации:

$$\frac{\partial f_i(x)}{\partial x_j} = \frac{f_i(x) - f_i(x_j^-)}{h} + O(h) \quad \text{или} \quad \frac{\partial f_i(x)}{\partial x_j} = \frac{f_i(x_j^+) - f_i(x_j^-)}{2h} + O(h^2),$$

где

$$x_j^{\pm} = x_1, \ldots, x_{j-1}, x_j \pm h, x_{j+1}, \ldots, x_n.$$

```
jacobian <- function(f,x,h=0.01,twoside=TRUE)
{
  n <- length(x)
  fc <- f(x)
  df <- array(dim=c(length(fc),n))
  for (i in 1:n)
  {
    if (twoside) # двухсторонняя аппроксимация
    {
      x[i] <- x[i]+h
      df[,i] <- f(x)
      x[i] <- x[i]-2*h
    }
    else          # односторонняя аппроксимация
    {
      df[,i] <- fc
      x[i] <- x[i]-h
    }
    df[,i] <- df[,i]-f(x)
    x[i] <- x[i]+h
  }
  if (twoside) df/(2*h) else df/h
}
```

Реализация метода Ньютона:

```
newton <- function(x0,f,eps=0.01,h=0.01,twoside=TRUE)
{
  newton_iter <- function(f)
     function(x) x-0.1*solve(jacobian(f,x,h,twoside),f(x))

  x <- x0
  repeat
  {
    x_next <- newton_iter(f)(x)
    if (max(abs(x-x_next))<eps && abs(f(x_next))<eps) return(x_next)
    x <- x_next
  }
}
```

**Пример.** *Решение системы уравнений*

$$x_1^2 + x_2^2 = 4,$$
$$(x_1 - 5)^2 + 2x_2^2 = 25.$$

```
f <- function(x) c(x[1]^2+x[2]^2-4,(x[1]-5)^2+2*x[2]^2-25)
x0 <- c(.5,.5)                          # первая пробная точка
```

Двухсторонняя аппроксимация производной:

```
newton(x0,f)                            # 0.7438359,  1.8538788
f(newton(x0,f))                         # -0.009841487,  -0.011334199
```

```
newton(x0,f,1e-5)                    # 0.744562, 1.856238
f(newton(x0,f,1e-5))                 # -9.398022e-6, -1.082298e-5

newton(x0,f,1e-5,1e-3)               # 0.744562, 1.856238
f(newton(x0,f,1e-5,1e-3))            # -9.398022e-6, -1.082298e-5
```

Односторонняя аппроксимация производной:

```
newton(x0,f,twoside=FALSE)           # 0.7438398, 1.8539368
f(newton(x0,f,twoside=FALSE))        # -0.009620572, -0.010937065

newton(x0,f,1e-5,1e-3,FALSE)         # 0.744562, 1.856238
f(newton(x0,f,1e-5,1e-3,FALSE))      # -9.355243e-6, -1.074678e-5
```

❑

### 2.5.2 Линейное программирование

Для решения задач линейного программирования (достаточно подробное описание линейных задач и методов их решения см., например, в [4])

$$c^{\mathsf{T}}x \to \text{extr}, \quad Ax \lessgtr b \tag{2.6}$$

применяется функция `solveLP` из библиотеки `linprog`:

```
solveLP( cvec, bvec, Amat, maximum = FALSE,
   const.dir = rep( '<=', length( bvec ) ),
   maxiter = 1000, zero = 1e-9, tol = 1e-6, dualtol = tol,
   lpSolve = FALSE, solve.dual = FALSE, verbose = 0 )
```

**Пример.** *Для выпуска двух типов продукции используется три вида сырья. Производственные коэффициенты (количество сырья, требующееся для производства единицы продукции), объёмы запасов и прибыль от продажи продукции приведены в таблице:*

| *Продукция* | *Сырьё* | | | *Прибыль* |
|---|---|---|---|---|
| *A* | *7* | *6* | *1* | *2* |
| *B* | *3* | *9* | *5* | *1* |
| *Объём запасов* | *60* | *80* | *35* | |

*Требуется, предполагая ненасыщенность рынка, составить такой план производства* $x = (x_1, x_2)$*, при котором прибыль будет максимальна.*

Задачу можно формально представить в стандартной форме задачи линейного программирования:

$$2x_1 + x_2 \to \max$$
$$\begin{cases} 7x_1 + 3x_2 & \leqslant 60 \\ 6x_1 + 9x_2 & \leqslant 80 \\ x_1 + 5x_2 & \leqslant 35 \\ x_1, x_2 \geqslant 0 \end{cases}$$

```
library(linprog)

Cvec <- c(2,1)                              # c
Bvec <- c(60,80,35)                         # b
Amat <- rbind( c(7,3), c(6,9), c(1,5))      # A

solveLP(Cvec,Bvec,Amat,maximum=TRUE)

Results of Linear Programming / Linear Optimization

Objective function (Maximum): 17.7778

Iterations in phase 1: 0
Iterations in phase 2: 2
Solution
      opt
1 6.66667
2 4.44444

Basic Variables
        opt
1   6.66667
2   4.44444
S 3 6.11111

Constraints
   actual dir bvec    free      dual dual.reg
1 60.0000  <=   60 0.00000 0.2666667 13.09524
2 80.0000  <=   80 0.00000 0.0222222 28.57143
3 28.8889  <=   35 6.11111 0.0000000  6.11111

All Variables (including slack variables)
        opt cvec     min.c     max.c       marg marg.reg
1   6.66667    2  0.666667 2.3333333         NA       NA
2   4.44444    1  0.857143 3.0000000         NA       NA
S 1 0.00000    0      -Inf 0.2666667 -0.2666667  13.0952
S 2 0.00000    0      -Inf 0.0222222 -0.0222222  28.5714
S 3 6.11111    0 -0.571429 0.0312500  0.0000000       NA
```

❑

### 2.5.3 Математическое программирование: скалярные функции

Функция optimize используется для поиска в заданном интервале экстремума функции $f: \mathbb{R} \to \mathbb{R}$:

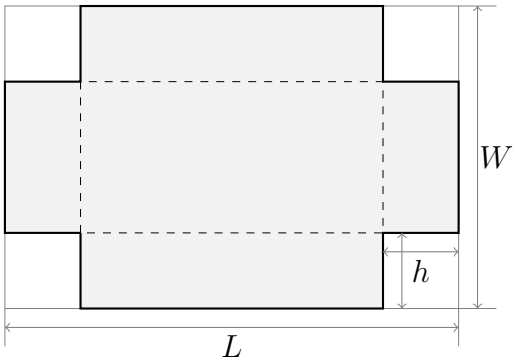
```
optimize(f, interval, ...,
         lower = min(interval), upper = max(interval),
         maximum = FALSE,
         tol = .Machine$double.eps^0.25)
```

Здесь

- `...` — значения дополнительных аргументов, передаваемые в функцию `f`.

В `optimize` применяется комбинация методов золотого сечения (устойчивость) и квадратичной интерполяции (скорость), последний из которых предполагает, что $f$ — непрерывная функция.

**Пример.** *Из прямоугольного бумажного листа размером $L \times W$ (например, $L = 100$ и $W = 50$) требуется вырезать уголки и склеить коробочку. Какова должна быть высота коробочки, чтобы её объём был максимален?*




Формальная постановка задачи:

$$V(h) = (L - 2h)(W - 2h)h \to \max$$

при условии $0 \leqslant h \leqslant \min(L, W)/2$.

```
V <- function(h,L,W) (L-2*h)*(W-2*h)*h

L <- 100; W <- 50
optimize(function(h) V(h,L,W),c(0,min(L,W)/2),maximum=TRUE)
```

Значение оптимальной высоты и максимального объёма:

```
$maximum
[1] 10.56624

$objective
[1] 24056.26
```

❑

**Пример** (взято из [15: с. 78]). *В структуре капитальных вложений на развитие химического завода важное место занимают затраты на приобретение и монтаж труб, а также затраты на установку насосного оборудования. Трубопровод длиной $L = 1000$ должен обеспечить подачу жидкости со скоростью $Q = 20$.*

*Требуется выбрать такой диаметр трубы $d$, чтобы затраты на приобретение труб, насосов и прокачивание жидкости были минимальны. Функция затрат описывается выражением*

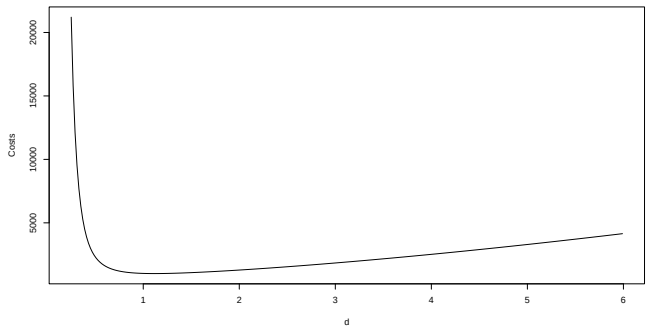
$$C(d) = 0.45L + 0.245Ld^{1.5} + 325h^{0.5} + 61.6h^{0.925} + 102,$$

*где*

$$h = 4.4 \times 10^{-8}\frac{LQ^3}{d^5} + 1.92 \times 10^{-9}\frac{LQ^{2.68}}{d^{4.68}}.$$

*Диаметр трубы должен находиться в диапазоне от 0.25 до 6.*

```
Costs <- function(d)
{
  L <- 1000
  Q <- 20

  h <- 4.4e-8*L*Q^3/d^5+1.92e-9*L*Q^2.68/d^4.68
  0.45*L+0.245*L*d^1.5+325*h^0.5+61.6*h^0.925+102
}

d <- seq(from=0.25,to=6,by=0.02)
plot(d,Costs(d),type='l',xlab='d',ylab='Costs')
```




```
optimize(Costs,c(.25,6))

$minimum
[1] 1.117262

$objective
[1] 1003.001
```

❑

Процедуру локализации точки минимума скалярной функции несложно реализовать самостоятельно. Решение разбивается на две части:

1. локализацию точки экстремума: определить интервал, содержащий хотя бы одну точку локального минимума;
2. уточнение решения: так уменьшить длину интервала, чтобы она стала меньше заданного значения.

Для определения исходного интервала применим метод Свенна [15: с. 51].

```
swann <- function(f,x0,delta=1e-3,maxiter=30,...)
{
  x <- c(NA,x0,NA)
  y <- c(NA,f(x0,...),NA)

  x[3] <- x[2]+delta      # делаем шаг в сторону, чтобы определить
  y[3] <- f(x[3],...)     # с какой стороны от x_0 находится точка минимума

  if (y[2] < y[3])        # если слева, то меняем направление на противоположное
  {
```

```
    x[1] <- x[3]
    y[1] <- y[3]
    delta <- -delta
  }

  counter <- 1
  repeat
  {
    x[3] <- x[2]+delta       # идём в выбранном направлении...
    y[3] <- f(x[3],...)
    delta <- delta*2         # ...каждый раз увеличивая длину шага...

    if (y[2] < y[3]) break   # ...пока значение функции не начнёт увеличиваться

    x[1:2] <- x[2:3]
    y[1:2] <- y[2:3]

    counter <- counter+1     # количество итерации должно быть ограничено
    if (counter>maxiter) stop('␣слишком␣много␣итераций')
  }

  c(min(x),max(x))
}
```

**Пример.** *Локализация интервала минимизации функции* $f(x) = (x - 1/3)^2$.

```
f <- function(x) (x-1/3)^2

swann(f,0)          # 0.127 0.511
swann(f,1)          # -0.023  0.745
swann(f,1/3)        # 0.3323333 0.3343333
```

❑

При неудаче, когда не удаётся за заданное число итераций найти подходящий интервал, выдаётся сообщение об ошибке:

```
swann(function(x) x,0)
```

```
Ошибка в swann(function(x) x, 0) : слишком много итераций
```

Существуют разные способы уменьшения длины интервала, локализующего точку минимума. Выберем один из самых простых, использующий сравнение значений функции в трёх точках (метод деления интервала пополам или метод трёх точек) [15: с. 52–53].

```
optim3p <- function(f,range,err=1e-3,...)
{
  L <- min(range)      # левая граница интервала
  R <- max(range)      # правая граница интервала
  m <- (L+R)/2         # середина интервала
  fm <- f(m,...)

  while (R-L > err)
  {
    xl <- (L+m)/2
```

```
    fl <- f(xl,...)
    if (fl < fm)      # если значение функции слева меньше, чем в центре
    {                 # то минимум находится в левой половине
       R <- m         # сдвигаем правую границу
       m <- xl
       fm <- fl
    }
    else              # иначе
    {
      xr <- (m+R)/2
      fr <- f(xr,...)
      if (fr < fm)    # если значение функции справа меньше, чем в центре
      {               # то минимум находится в правой половине
         L <- m       # сдвигаем левую границу
         m <- xr
         fm <- fr
      }
      else            # в противном случае минимум находится в «средней» половине
      {               # сдвигаем обе границы
         L <- xl
         R <- xr
      }
    }
  }

  list(x.opt=(L+R)/2,f.opt=f((L+R)/2,...))
}
```

**Пример** (продолжение). *Поиск точки минимума функции функции* $f(x) = (x = 1/3)^2$.

```
optim3p(f,swann(f,0))
```

```
$x.opt
[1] 0.33325
$f.opt
[1] 6.944444e-09
```

❑

**Пример** (продолжение примера на с. 68). *Минимизация затрат.*

```
optim3p(Costs,c(.25,6))
```

```
$x.opt
[1] 1.117203
$f.opt
[1] 1003.001
```

❑

### 2.5.4 Математическое программирование: функции многих аргументов

Для безусловной минимизации функции многих переменных $f: \mathbb{R}^m \to \mathbb{R}$ $(m > 1)$ используется функция `optim`:

```
optim(par, fn, gr = NULL, ...,
      method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN", "Brent"),
      lower = -Inf, upper = Inf,
      control = list(), hessian = FALSE)
```

Здесь

- `par` — первая пробная точка.
- `fn` — минимизируемая функция.

  Вообще говоря, функция `fn` может возвращать значения `NA` и `NaN` (за исключением `L-BFGS-B`, где значения `fn` должны быть всегда конечны).

  Значение `fn` в `par` должно быть конечно.
- `gr` — функция, вычисляющая вектор градиента (используется в методах `BFGS`, `CG` и `L-BFGS-B`).

  Если `gr` не задана, то для оценки градиента применяется конечно-разностная аппроксимация.
- `...` — дополнительные параметры, передаваемые в `fn` и `gr`.
- `method` — используемый метод численной минимизации.
  - `Nelder-Mead` — модификация Нелдера-Мида поиска по симплексу. Устойчивый, но относительно медленный.

    Метод нулевого порядка, то есть использует только значения `fn` и поэтому может применяться для минимизации недифференцируемых функций.

    Описание см. в [15: с. 87–93].
  - `BFGS` — квазиньютоновский метод Бройдена, Флетчера, Голдфарба и Шанно. Используя значения функции и градиента, аппроксимирует значение второй производной, что повышает скорость сходимости при удачно выбранной начальной пробной точке.

    Описание см. в [15: с. 128].
  - `CG` — метод, основанный на методе сопряжённых градиентов Флетчера-Ривса. Менее устойчив, по сравнению с `BFGS`, но, так как не сохраняет матрицу вторых производных, может работать с задачами большей размерности.

    Описание см. в [15: с. 120–123].
  - `L-BFGS-B` — для каждой переменной задаются нижняя и верхняя границы. Первое пробное значение должно удовлетворять заданным ограничениям.
  - `SANN` — метод имитации отжига. Так как использует только значения функции, то относительно медленный (как и многие методы из теории слабого искусственного интеллекта), но может использоваться для работы с недифференцируемыми и мультимодальными функциями.

    Описание см., например, в [7: с. 25–41].

– `Brent` — использует `optimize` и работает со скалярными функциями.

- `control` — список, позволяющий более точно настроить поведение `optim`.
- `hessian` — симметричная матрица, оценка матрицы Гессе найденного решения.

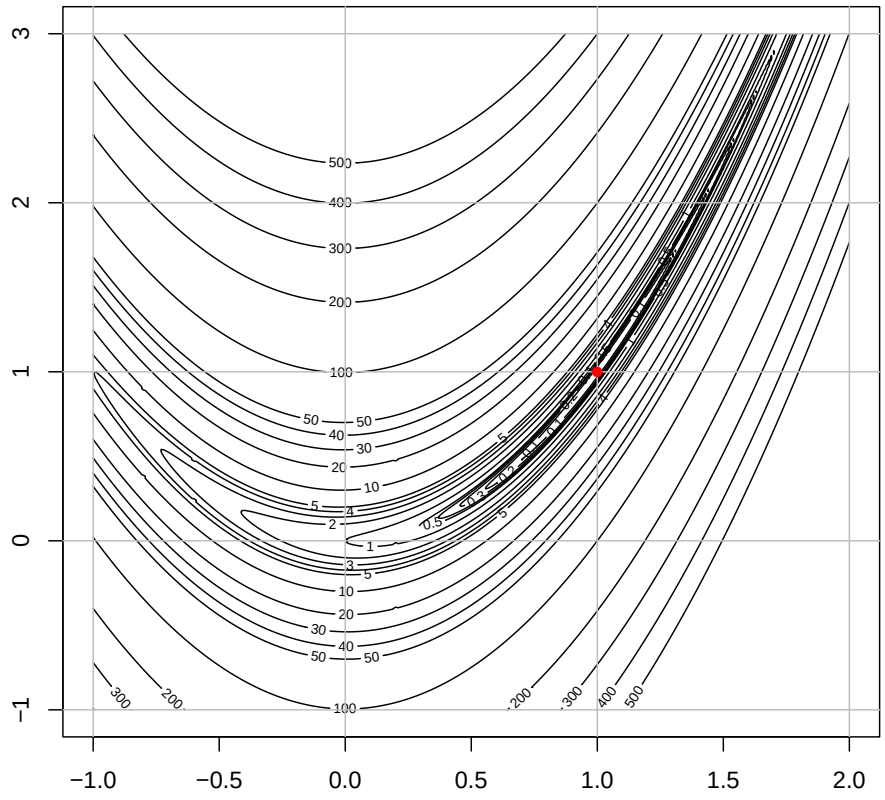
**Пример.** *Поиск минимума функции Розенброка (точное решение:* $x^\star = (1,1)$*)*

$$f(x_1, x_2) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2.$$

Функция Розенброка

```
f <- function(x) 100*(x[2]-x[1]^2)^2+(1-x[1])^2
```

является примером функции с сильной овражностью (когда значения градиента в разных направлениях могут существенно, на порядки отличаются друг от друга):




Если начальная точка выбрана достаточно близко к решению, то все методы дают более или менее близкие приближения:

```
optim(c(0,0),f)                    # 0.9999564, 0.9999085
optim(c(0,0),f,method='BFGS')      # 0.9998000, 0.9996001
optim(c(0,0),f,method='SANN')      # 0.9983640, 0.9961739
```

Однако если начальный выбор точки неудачен, то результаты могут существенно различаться:

```
optim(c(-10,-10),f)                # 0.4425542, 0.1711704 плохо
optim(c(-10,-10),f,method='BFGS')  # −3.331027, 11.037737 ужасно
optim(c(-10,-10),f,method='SANN')  # 1.000834, 1.001517 хорошо
```

❑

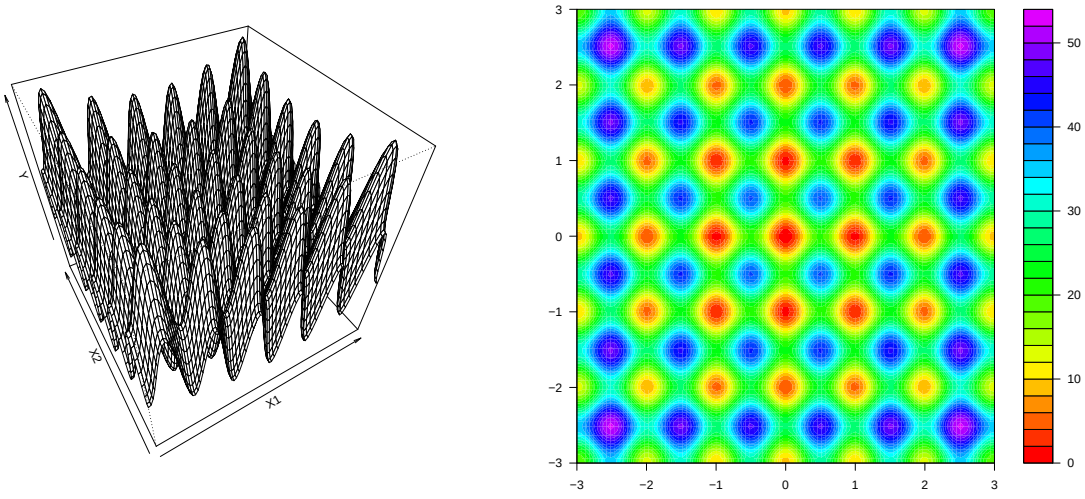
**Пример.** *Поиск глобального минимума мультимодальной функции Растригина*

$$f(x \mid A) = An + \sum_{i=1}^{n}\bigl(x_i^2 - A\cos(2\pi x_i)\bigr).$$

Функция Растгигина

```
f <- function(x,A=10) A*length(x)+sum(x^2-A*cos(2*pi*x))
```

имеет один глобальный минимум в точке $(0, 0)$ и множество минимумов локальных:




Так как функция Растригина мультимодальна, то при неудачном выборе первой пробной точки решением будет точка *локального* минимума. Если выполнить `optim` несколько раз, когда параметр `method` имеет значение `SANN` (или с различными начальными точками), то есть вероятность найти *глобальный* минимум.

```
x0 <- runif(2,min=-2,max=2)   # Повезёт или не повезёт?
optim(x0,f)
optim(x0,f,method='BFGS')
optim(x0,f,method='SANN')
```

❑

*Замечание.* При поиске *глобального* экстремума мультимодальной функции лучше использовать методы оптимизации, разработанные в рамках теории слабого искусственного интеллекта, такие как *генетическое программирование* (см. раздел 2.5.5).

### 2.5.5 Генетические алгоритмы

Решение оптимизационных задач с помощью генетических алгоритмов реализовано в функции `rbga` из библиотеки `genalg`:

```
rbga(stringMin=c(), stringMax=c(),
     suggestions=NULL,
     popSize=200, iters=100,
     mutationChance=NA,
     elitism=NA,
     monitorFunc=NULL, evalFunc=NULL,
     showSettings=FALSE, verbose=FALSE)
```

Здесь

- `stringMin` и `stringMax` — минимальные и максимальные значения генов.
- `suggestinos` — список предлагаемых хромосом.
- `popSize` — объём популяции.
- `iters` — число поколений.
- `mutationChance` — шанс мутации. По умолчанию `1/(popSize+1)`.
- `elitism` — число хромосом, которые будут сохранены в следующей популяции. По умолчанию $\approx 20\%$.
- `monitorFunc` — процедура, запускаемая на каждой итерации.
- `evalFunc` — целевая функция (функция оценки приспособленности хромосомы).

Описание генетических алгоритмов см. в [7: с. 112–140].

**Пример.** *Минимизация мультимодальной функции Растригина*

$$f(x \mid A) = An + \sum_{i=1}^{m}\bigl(x_i^2 - A\cos(2\pi x_i)\bigr).$$

Функция Растригина

```
f <- function(x,A=5) sum(x^2)+A*(length(x)-sum(cos(2*pi*x)))
```

имеет один глобальный минимум в точке $(0, 0, \ldots, 0)$ и множество локальных минимумов.

Вообще говоря, функция Растригина имеет $m$ аргументов: $f\colon \mathbb{R}^m \to \mathbb{R}$. Для простоты примем $m = 2$ и будем искать решение на области $[-10, 10] \times [-10, 10]$. Исходные данные:

```
xmin <- -10
xmax <-  10
```

Функция `monitor` будет использоваться для графического отображения хода решения:

```
n <- 250
X <- seq(from=xmin,to=xmax,length=n)   # f:[-10,10]x[-10,10]->R
z <- array(dim=c(n,n))
for (i in 1:n) for (j in 1:n) z[i,j] <- f(c(X[i],X[j]))

monitor <- function(obj)
{
  contour(X,X,z,xlim=c(xmin,xmax),ylim=c(xmin,xmax),
          xlab=expression(x[1]),ylab=expression(x[2]))
  points(obj$population,pch=16,col='blue')
}
```

Поиск минимума с графическим отображением прогресса:

```
library(genalg)
res <- rbga(rep(xmin,2),rep(xmax,2),
            monitorFunc=monitor,
            evalFunc=f,mutationChance=0.01)
```

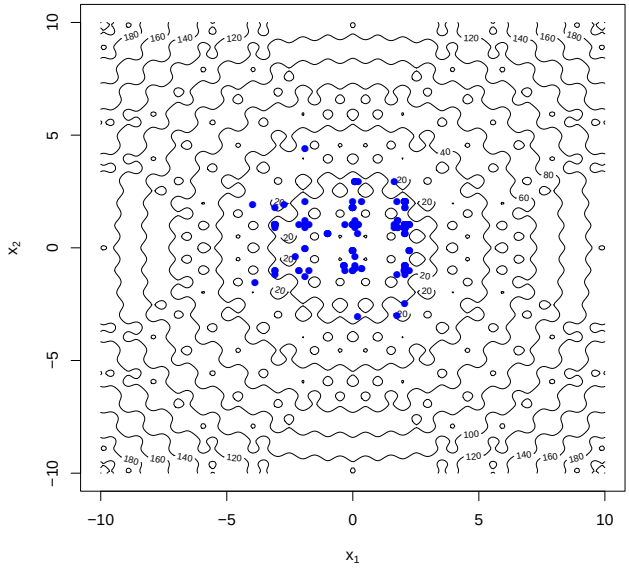
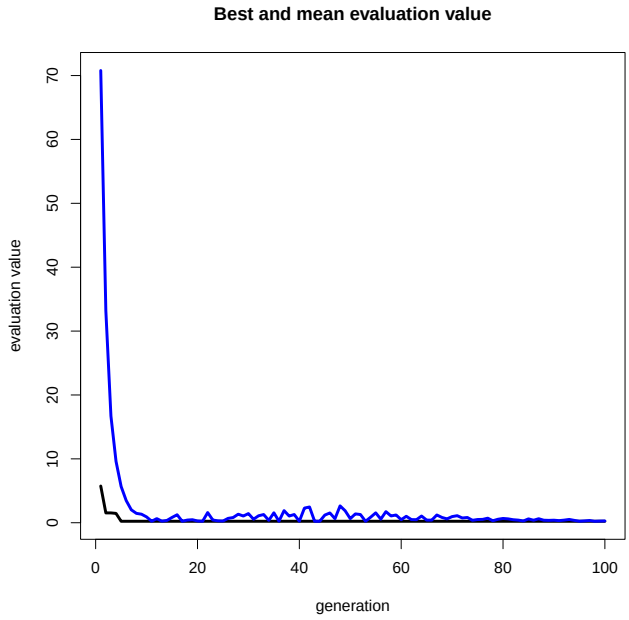



График динамики изменений наилучших и средних значений по популяции:

```
plot(res,lwd=3)
```




Оптимизация без использования функции мониторинга:

```
res <- rbga(rep(xmin,2),rep(xmax,2),
            evalFunc=f,mutationChance=0.01)

list(x.opt = res$population[1,], f.opt = f(res$population[1,]))

$x.opt
[1]  0.03175555 -0.03606510

$f.opt
[1] 0.2293307
```

❑

### 2.5.6 Условная оптимизация

Функция `constrOptim` позволяет решить задачу минимизации целевой функции при наличии линейных ограничений-неравенств:

```
constrOptim(x0, f, grad, A, b, mu = 1e-04, control = list(),
            method = if(is.null(grad)) "Nelder-Mead" else "BFGS",
            outer.iterations = 100, outer.eps = 1e-05, ...,
            hessian = FALSE)
```

Здесь

- `x0` — начальная пробная точка;
- `f` — минимизируемая функция;
- `grad` — функция, вычисляющая вектор градиента, или `NULL`;
- `A` и `b` — матрица и вектор, определяющие систему ограничений:

$$Ax - b \geqslant 0.$$

**Пример.** *Требуется найти объём потребления* $(x_1, x_2)$*, дающего наибольшее значение функции полезности*

$$U(x) = x_1^{0.7} x_2^{0.3}$$

*при наличии бюджетного ограничения*

$$5x_1 + 2x_2 \leqslant 100, \quad x_1, x_2 \geqslant 0.$$

```
U <- function(x) x[1]^0.7*x[2]*0.3
g <- function(x) 100-(5*x[1]+2*x[2])

constrOptim(c(0,0),
            function(x) -U(x),  # U(x) → max
            NULL,               # функция для градиента не задана
            c(-5,-2),           # A
            -100)               # b


$par
[1]  8.234075 29.414813

$value
[1] -38.6027
.....................
```

❑

Также для решения задачи условной оптимизации можно использовать функцию `solnp` из библиотеки `Rsolnp`:

```
solnp(pars, fun, eqfun = NULL, eqB = NULL, ineqfun = NULL, ineqLB = NULL,
      ineqUB = NULL, LB = NULL, UB = NULL, control = list(), ...)
```

Для решения задачи условной нелинейной оптимизации

$$f(x) \to \min$$
$$\begin{cases} h(x) = 0 \\ l_h \leqslant g(x) \leqslant u_h \\ l_x \leqslant x \leqslant u_x \end{cases}$$

применяется расширенный метод множителей Лагранжа.

**Пример** (продолжение)**.**

```
library(Rsolnp)

solnp(c(1,1),
      fun=function(x)-U(x),      # U(x) → max
      ineqfun=g,                 # 0 ⩽ g(x) ⩽ 100
      ineqLB=0,ineqUB=100,
      LB=c(0,0))                 # x ⩾ 0


.......................
$pars
[1]  8.235294 29.411765
.......................
```

❑

**Штрафные функции** используются при сведéнии решения задачи минимизации функции $f$

$$f(x) \to \min$$

при наличии ограничений на решение в виде равенств и/или неравенств

$$h(x) = 0, \quad g(x) \geqslant 0$$

к задаче безусловной минимизации

$$F(t \mid q_h, q_g) = f(t) + q_h P_h\big(h(t)\big) + q_g P_g\big(g(t)\big) \to \min.$$

Здесь $P_h$ и $P_g$ — функции, накладывающие штрафы за нарушения ограничений равенств и неравенств соответственно. Коэффициенты $q_h$ и $q_g$ регулируют влияние штрафов за нарушение ограничений.

*Замечание.* Не следует сразу задавать большие значения масштабирующих штрафов для коэффициентов $q_h$ и $q_g$. Лучше сначала присвоить им относительно небольшие значения, а затем постепенно увеличивать, используя в качестве начальной пробной точки решение, полученное на предыдущем шаге.

**Пример** (продолжение)**.**

- Ограничение-равенство:

$$U(x) \to \max, \quad g(x) = 100.$$

```
Fh <- function(x,q) -U(x)+q*(g(x)^2)

x <- c(1,1)
for (q in c(0.1,1,10,10^3,10^10)) optim(x,Fh,NULL,q)$par -> x

x                          # 8.22923 29.42693
c(U(x),g(x))               # 3.860269e+01 -5.379789e-08
```

- Ограничения-неравенства:

$$U(x) \to \max, \quad 100 - g(x) \geqslant 0, \quad x \geqslant 0.$$

```
Fg <- function(x,q)
{
  gh <- g(x)

  -U(x) + q*(ifelse(gh<0,gh^2,0) + sum(x[x<0]^2))
}

x <- c(1,1)
for (q in c(0.1,1,10,10^3,10^10)) optim(x,Fg,NULL,q)$par -> x

x                          # 8.233651 29.415864
c(U(x),g(x))               # 3.860269e+01 1.435124e-05
```

❑

# Глава 3

# Программирование

## 3.1 Управляющие конструкции

Для управления выполнением действий применяются условные выражения и циклы.

### 3.1.1 Условные выражения

**Условие `if`.** Если `условие` истинно, то выполняется `действие`.

```
if (условие) действие
```

```
X <- rnorm(100)
for (x in X)
  if (abs(x)>3) message('x␣выходит␣за␣пределы␣трёх␣сигм')
```

Если `условие` истинно, то выполняется `действие1`, иначе выполняется `действие2`.

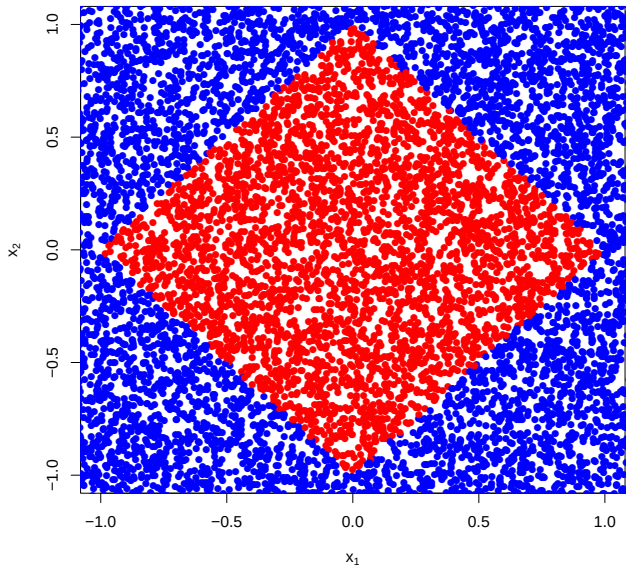
```
if (условие) действие1 else действие2
```

**Пример.** *Генерирование случайных точек* $(\xi_1, \xi_2)$, $\xi_1, \xi_2 \sim \mathcal{U}(-1, 1)$ *с последующим выбором цвета:*

$$Цвет(x_1, x_2) = \begin{cases} красный, & |x_1| + |x_2| \leqslant 1, \\ синий, & |x_1| + |x_2| > 1. \end{cases}$$

```
plot(c(-1,1),c(-1,1), type='n',
     xlab=expression(x[1]),ylab=expression(x[2]))

n <- 10000
x <- matrix(runif(2*n,min=-1.2,max=1.2),nrow=n,ncol=2)

for (i in 1:n)
{
  if (sum(abs(x[i,]))<=1)
    color = 'red'
  else
    color = 'blue'

  points(x[i,1],x[i,2],pch=16,col=color)
}
```

Результат — красный ромб на синем фоне:




❑

Если действия при выполении и невыполнении условия представлены единственными выражениями, то условный оператор можно заменить функцией `ifelse`:

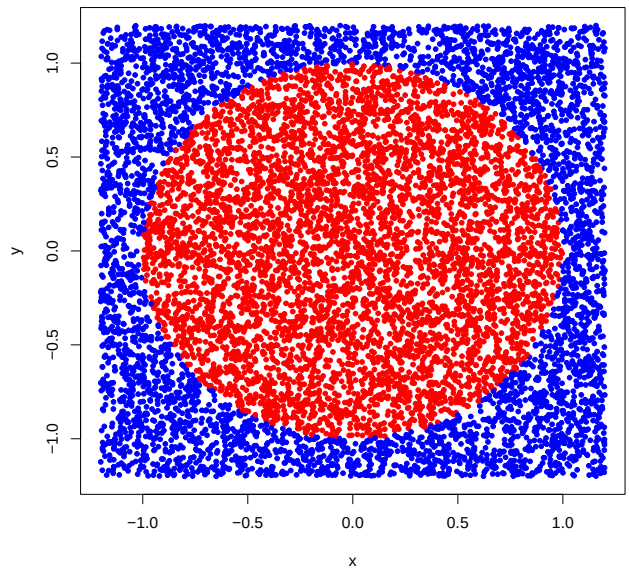
```
ifelse(условие, выражение-если-TRUE, выражение-если-FALSE)
```

**Пример.** *Генерирование случайных точек* $(\xi_1, \xi_2)$, $\xi_1, \xi_2 \sim \mathcal{U}(-1.3, 1.3)$ *с последующим выбором цвета:*

$$\textit{Цвет}(x_1, x_2) = \begin{cases} \textit{красный}, & x_1^2 + x_2^2 \leqslant 1, \\ \textit{синий}, & x_1^2 + x_2^2 > 1. \end{cases}$$

```
n <- 10000
x <- runif(n,min=-1.2,max=1.2)
y <- runif(n,min=-1.2,max=1.2)
plot(x,y,pch=20,
     col=ifelse(x^2+y^2<=1,'red','blue'))
```

Результат — красный круг на синем фоне:



❑

**Пример.** *Реализация функции Хевисайда (единичная ступенька):*

$$\chi(x) = \begin{cases} 0, & x < 0, \\ 1, & \textit{иначе.} \end{cases}$$

```
heaviside <- function(x) ifelse(x<0,0,1)
```

❑

**Пример.** *Реализация кусочно-линейной функции (piecewise linear)*

$$pwlin(x) = \begin{cases} -1, & x \leqslant -1, \\ x, & -1 \leqslant x \leqslant 1, \\ 1, & x \geqslant 1 \end{cases}$$

Больше двух вариантов, поэтому используется вложение функций `ifelse`

```
pslin <- function(x) ifelse(x<=-1,-1,ifelse(x>=1,1,x))
```

❑

**Выбор `switch`** Функция `switch` вычисляет значение `EXPR` и выводит соответствующий элемент из `...`.

```
switch(EXPR, ...)
```

**Пример.** *Реализация функции, вычисляющей расстояние (евклидово, манхэттенское или Чебышёва) между точками $x, y \in \mathbb{R}^m$:*

```
distance <- function(x,y,type='euclidean')
  switch(type,
    euclidean = sum((x-y)^2)^0.5,   # sqrt(sum_{i=1}^n (x_i - y_i)^2)
    manhattan = sum(abs(x-y)),      # sum_{i=1}^n |x_i - y_i|
    chebyshev = max(abs(x-y)))      # max_{i=1,...,n} |x_i - y_i|

x <- rnorm(10)                      # x, y ∈ R^10
y <- rnorm(10)

distance(x,y)                       # аналогично distance(x,y,'euclidean')
distance(x,y,'manhattan')
distance(x,y,'chebyshev')
```

❑

### 3.1.2 Циклы

**Цикл с заданным числом итераций:** индексная `переменная` пробегает все элементы `последовательности`

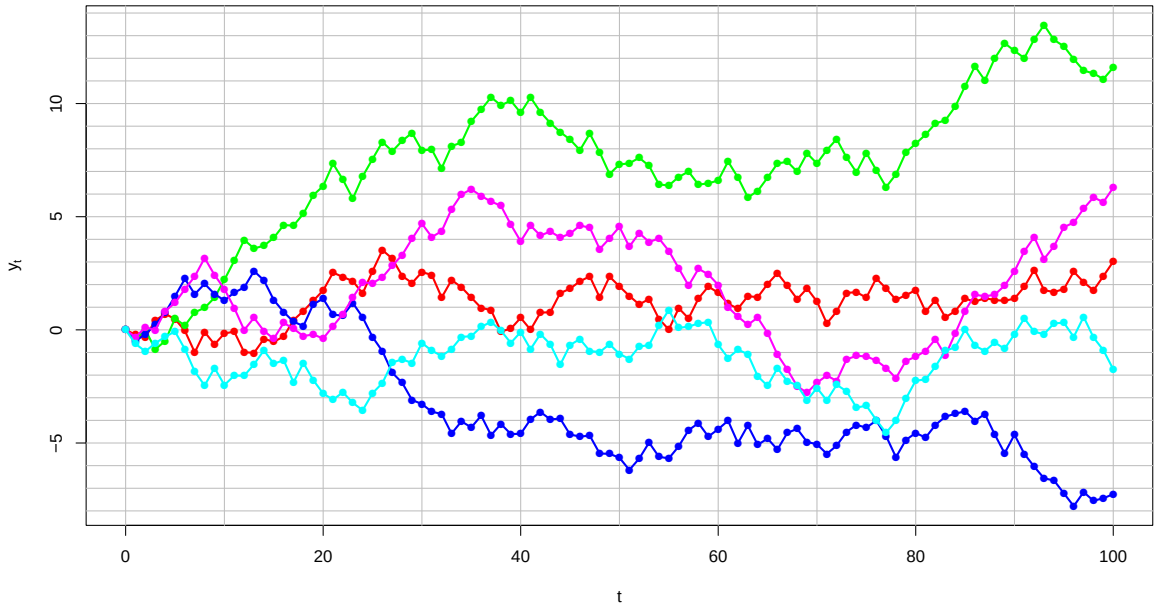
```
for (переменная in последовательность) выражение
```

**Пример.** *Случайное блуждание:*

$$y_{i+1} = y_i + \varepsilon_i, \qquad \varepsilon_i \sim \mathcal{U}(-1, 1).$$

Несколько ($m$) реализаций процесса случайного блуждания:

```
n <- 100     # t = 0,1,...,n
m <- 5

# место для хранения
y <- matrix(nrow=m,ncol=n+1)
# нулевые начальные значения
y[,1] <- rep(0,m)

# реализация процесса случайного блуждания
for (i in 1:n) y[,i+1] <- y[,i]+runif(m,min=-1,max=1)

# чистый холст для рисования
plot(c(0,n),c(min(y),max(y)),type='n',
     xlab='t',ylab=expression(y[t]))
# сетка
for (i in floor(min(y)):ceiling(max(y))) abline(h=i,col='grey')
for (i in seq(from=0,to=n,by=10)) abline(v=i,col='grey')
# разные цвета для разных реализаций
color <- c('red','green','blue','magenta','cyan')
# m реализаций процесса
for (i in 1:m) lines(0:n,y[i,],type='o',pch=16,col=color[i],lwd=2)
```




❑

**Цикл с предусловием** выполняется, пока `условие` истинно:

```
while (условие) выражение
```

**Повторение** потенциально может выполняться бесконечно (ограничение находится в теле цикла):

```
repeat выражение
```

**Управление циклом.** Для перехода к следующей итерации без выполнения оставшейся части тела цикла используется команда `next`. Команда `break` применяется для выхода из цикла.

**Пример.** *Вычисление факториала:*

$$n! = 1 \times 2 \times \cdots \times n, \qquad n \in \mathbb{N},$$
$$0! = 1 \quad \textit{по определению.}$$

```
n <- 10

# цикл с предусловием
fact <- 1; i <- 1
while (i <= n)
{
  fact <- fact*i
  i <- i+1
}
fact

# цикл с постусловием
fact <- 1; i <- 1
repeat
{
  fact <- fact*i
  i <- i+1
  if (i > n) break
}
fact
```

Вместо цикла можно использовать рекурсию:

```
fact <- function(n)
{
  if (n==0 || n==1) return(1)
  else return(n*fact(n-1))
}

for (i in 0:10) cat(format(i,width=2,justify='right'),
                    '␣÷␣',fact(i),'\n')
```

```
 0  ÷  1
 1  ÷  1
 2  ÷  2
 3  ÷  6
 4  ÷  24
 5  ÷  120
 6  ÷  720
 7  ÷  5040
 8  ÷  40320
 9  ÷  362880
10  ÷  3628800
```

❑

**Пример.** *Оценка площади единичного круга методом Монте-Карло (метод случайных испытаний).*

Сначала оценим вероятность того, что случайная точка $x = (\xi_1, \xi_2)$, где $\xi_1, \xi_2 \sim \mathcal{U}(-1, 1)$, попадёт внутрь единичного круга:

```
N <- 100000 # количество испытаний

S <- 0
i <- 0
repeat
{
  x <- runif(2,min=-1,max=1)
  if (sum(x^2)<=1) S <- S+1
  i <- i+1
  if (i >= N) break
}
prob <- S/N
```

Площадь круга равна произведению вероятности попасть внутрь круга на площадь квадрата, в котором находится круг:

```
prob*4
```

*Замечание.* Вообще говоря, часто можно обойтись без циклов, используя способность R напрямую работать с векторами и матрицами:

```
N <- 100000
x <- runif(N,min=-1,max=1)
y <- runif(N,min=-1,max=1)
S <- sum(x^2+y^2 <= 1)
```

Правда, в таком случае требуется хранить в рабочей памяти промежуточные значения x и y. Вместо этого можно объединить несколько операций в одном выражении:

1. создаём матрицу, содержащую случайные координаты пробных точек;
2. применяем сложение по столбцам;
3. проверям выполнение условий;
4. вычисляем количество успешных опытов.

```
S <- sum(apply(matrix(runif(2*N,min=-1,max=1)^2,nrow=2,ncol=N),
               2,sum) <= 1)
```

Оценка площади круга:

```
S/N *4
```

❑

## 3.2 Функции

С точки зрения парадигмы структурного программирования, использование функций (подпрограммы, возвращающие результат вычислений) и процедур (подпрограммы, выполняющие определённые действия) позволяет упростить структуру программы. С точки зрения

функционального подхода, функции могут использоваться как аргументы других функций (функций высших порядков), а также быть результатами вычислений.

R, в принципе, поддерживает как императивный структурный, так и функциональный и объектно-ориентированный стили программирования. Функции и процедуры определяются в нём выражением `function`:

```
function(аргументы) выражение
```

или (сокращённая форма)

```
\(аргументы) выражение
```

Возвращается результат последнего вычисленного выражения. Также результат можно вернуть с помощью `return`.

**Пример** (Рекурсия vs. итерация)**.**

Для определения значения $i$-го числа Фибоначчи $f_i$ можно использовать рекуррентное выражение

$$f_0 = 0,\ f_1 = 1, \qquad f_n = f_{n-1} + f_{n-2}, \quad n = 2, 3, \ldots \tag{3.1}$$

Если просто переписать (3.1) в виде рекурсивной функции

```
fib <- function(n)
  switch(as.character(n),
    '0' = 0,
    '1' = 1,
    fib(n-1)+fib(n-2))
```

то получится функция, использующая *древовидную* рекурсию (см. [1: стр. 53–55]), которая прекрасно выглядит в виде математического выражения, но *ужасно* неэффективна в плане вычислений:

```
for (i in seq(from=0,to=30,by=10))
{
  start <- Sys.time()
  fib(i)
  cat(format(i,width=2),':',Sys.time()-start,'сек.\n')
}
```

```
 0 : 5.292892e-05 сек.
10 : 0.001464844 сек.
20 : 0.02508593 сек.
30 : 3.083133 сек.
```

Использование аккумулятора позволяет преобразовать древовидную рекурсию в линейную хвостовую или, другими словами, рекурсивный процесс в итерационный:

```
fib <- function(n)
{
  fib_iter <- function(a,b,count)
    ifelse(count==0,b,fib_iter(a+b,a,count-1))

  fib_iter(1,0,n)
}
```

В этом случае объём вычислений будет пропорционален $n$:

```
 0 : 4.792213e-05 сек.
10 : 0.001714945 сек.
20 : 0.0001108646 сек.
30 : 0.0001499653 сек.
```

Конечно, итерационный процес можно записать явно:

```
fib <- function(n)
{
  if (n==0) return(0)
  else if (n==1) return(1)

  a <- 0; b <- 1
  for (i in 0:n)
  {
    tmp <- a+b
    b <- a
    a <- tmp
  }
  b
}
```

но такая запись будет несколько длиннее (хотя, возможно, и более понятна).

❑

**Пример** (поиск в пространстве состояний)**.** *Решение известной задачи о перевозке крестьянином через реку волка, козы и капусты.*

Описание используемых далее алгоритмов поиска в пространстве состояний задачи см. в [27].

**Поиск в глубину** проход по ветке до достижения целевого состояния или тупика. В последнем случае — переход на другую ветку.

```
# матрица для хранений сгенерированных состояний
# строка матрицы определяет положение
# (крестьянин,волк,коза,капуста)
# 0 — левый берег, 1 — правый
states <- matrix(c(0,0,0,0),nrow=1,ncol=4)
# целевое состояние
goal <- c(1,1,1,1)
# индексы (номера строк) ещё не обработанных состояний
work <- 1
# индексы состояний, составляющих решение задачи
solution <- c()

# функции, выделяющие из состояния расположение
peasant <- function(s) s[1]   # крестьянина
wolf    <- function(s) s[2]   # волка
goat    <- function(s) s[3]   # козы
cabbage <- function(s) s[4]   # капусты

# проверка состояния на допустимость
is_available <- function(s)
{
  # недопустимо: волк и коза на одном берегу, крестьянин — на другом
```

```
  d1 <- peasant(s)!=wolf(s) && wolf(s)==goat(s)
  # недопустимо: коза и капуста на одном берегу, крестьянин — на другом
  d2 <- peasant(s)!=goat(s) && goat(s)==cabbage(s)
  # если состояние не является недопустимым, то оно допустимо
  !(d1 || d2)
}

# проверка, не является ли состояние s уже сгенерированным
exists <- function(s)
{
  for (i in 1:dim(states)[1])
    if (all(states[i,]==s)) return(TRUE)

  FALSE
}

# определение берега после перевозки
move <- function(side) ifelse(side==0,1,0)

# обработка нового состояния s
# счётчик counter содержит количество новых состояний,
# помещённых в список states
process <- function(s, counter)
{
  if (is_available(s) && !exists(s))
  {
    states <<- rbind(states,s)
    work <<- c(dim(states)[1],work)
    counter+1
  }
  else counter
}

# определение состояний-потомков для состояния s
generate <- function(s)
{
  counter <- 0
  side <- move(peasant(s))

  # крестьянин плывёт один
  counter <- process(c(side,wolf(s),goat(s),cabbage(s)), counter)
  # крестьянин перевозит волка
  if (peasant(s)==wolf(s))
  {
    counter <- process(c(side,side,goat(s),cabbage(s)),counter)
  }
  # крестьянин перевозит козу
  if (peasant(s)==goat(s))
  {
    counter <- process(c(side,wolf(s),side,cabbage(s)),counter)
  }
  # крестьянин перевозит капусту
  if (peasant(s)==cabbage(s))
```

```
  {
    counter <- process(c(side,wolf(s),goat(s),side),counter)
  }

  counter!=0
}

# трассировка решения
trace <- TRUE

# будем (потенциально бесконечно) решать задачу, пока не найдём решение
# или не докажем, что решения не существует
while(TRUE)
{
  # рабочий список пуст — нет решения
  if (length(work)==0) stop('решение не найдено...', call.=FALSE)
  # обрабатываемое состояние совпадает с целевым — задача решена
  if (all(states[work[1],]==goal))
  {
    solution <- c(solution, work[1])
    break
  }

  # индекс обрабатываемого состояния
  idx <- work[1]
  work <- work[-1]
  solution <- c(solution,idx)
  if (trace) cat('в решение добавляется ',idx,'\n')

  if (!generate(states[idx,]))
  {
    # если тупик, то возвращаемся на один шаг назад
    n <- length(solution)
    if (n==1)
      solution <- c()
    else
      solution <- solution[-n]

    if (trace) cat('из решения удаляется последнее состояние\n')
  }

  if (trace)
  {
    cat('решение                ',solution,'\n')
    cat('рабочий список         ',work,'\n')
    cat('обработанное состояние',states[idx,],'\n')
    cat('следующее в обработке ',states[work[1],],'\n\n')
  }
}

# выводим найденное решение
for (s in solution) cat(states[s,],'\n')
```

```
в решение добавляется  1
```

```
решение                1
рабочий список         2
обработанное состояние 0 0 0 0
следующее в обработке  1 0 1 0

...

в решение добавляется  7
решение                1 2 3 5 6 7
рабочий список         9 8 4
обработанное состояние 1 1 0 1
следующее в обработке  0 1 0 0

в решение добавляется  9
из решения удаляется последнее состояние
решение                1 2 3 5 6 7
рабочий список         8 4
обработанное состояние 0 1 0 0
следующее в обработке  0 1 0 1

в решение добавляется  8
решение                1 2 3 5 6 7 8
рабочий список         10 4
обработанное состояние 0 1 0 1
следующее в обработке  1 1 1 1

0 0 0 0
1 0 1 0
0 0 1 0
1 0 1 1
0 0 0 1
1 1 0 1
0 1 0 1
1 1 1 1
```

**Поиск в ширину** проверка всех состояний на одном уровне, затем переход на следующий уровень. В отличие от поиска в глубину, где путь от начального состояния до цели сохраняется в переменной `solution`, при поиске в ширину каждое состояние должно хранить индекс своего предка (у начального состояния — $-1$).

```
# матрица для хранений сгенерированных состояний
# строка матрицы определяет положение объектов и индекс предыдущего состояния
# (крестьянин,волк,коза,капуста, предок)
# 0 — левый берег, 1 — правый
states <- matrix(c(0,0,0,0,-1),nrow=1,ncol=5)
goal <- c(1,1,1,1)
work <- 1

peasant <- function(s) s[1]
wolf    <- function(s) s[2]
goat    <- function(s) s[3]
```

```
cabbage <- function(s) s[4]
prev    <- function(s) s[5]   # индекс предыдущего состояния

# is_awailable и move те же, что и при поиске в глубину
is_awailable <- function(s) { ... }
move <- function(side) { ... }

# проверка на существование учитывает, что последний элемент
# в состоянии — индекс состояния-предка
exists <- function(s)
{
  for (i in 1:dim(states)[1])
    if (all(states[i,-5]==s[-5])) return(TRUE)

  FALSE
}

# так как нет возвратов, то счётчик не используется
process <- function(s)
{
  if (is_awailable(s) && !exists(s))
  {
    states <<- rbind(states,s)
    work <<- c(work,dim(states)[1])
  }
}
generate <- function(idx)
{
  s <- states[idx,]
  side <- move(peasant(s))

  process(c(side,wolf(s),goat(s),cabbage(s),idx))

  if (peasant(s)==wolf(s))
    process(c(side,side,goat(s),cabbage(s),idx))

  if (peasant(s)==goat(s))
    process(c(side,wolf(s),side,cabbage(s),idx))

  if (peasant(s)==cabbage(s))
    process(c(side,wolf(s),goat(s),side,idx))
}

while(TRUE)
{
  # если рабочий список пусть, то решения не существует
  if (length(work)==0) stop('решение не найдено...', call.=FALSE)
  # целевое решений достигнуто, задача решена
  if (all(states[work[1],-5]==goal)) break

  idx <- work[1]
  work <- work[-1]
```

```
  generate(idx)
}

# формируем путь от начального состояния к целевому
solution <- work[1]
while (prev(states[solution[1],]) != -1)
  solution <- c(prev(states[solution[1],]),solution)

# печатаем решение
for (s in solution) cat(states[s,-5],'\n')
```

❑

**Пример** (направленный поиск: алгоритм $A^\star$). *На левом берегу реки находятся $N$ семейных пар (мужчины и женщины), которые хотят переправиться на правый берег. Посередине реки расположен остров. Грести могут как мужчины, так и женщины.*

*Недопустимое состояние: женщина не может находиться в компании других мужчин, если её мужчины нет рядом.*

```
N <- 4 # размерность задачи (количество пар)

# 0 — левый берег
# 1 — правый берег
# 2 — остров
sides <- c(0,1,2)

# целевое состояние
goal <- c(rep(1,2*N))

# оценка расстояния от начала до цели через текущее состояние
dist <- function(s) sum(s[1:(2*N)]!=goal)+s[2*N+3]

# 1:N мужчины
# (N+1):(2*N) женщины
# 2N+1 лодка
# 2N+2 предок
# 2N+3 глубина состояния (количество шагов от начального состояния)
start  <- c(rep(0,2*N+1),-1,0)
# 2N+4 оценка расстояния от начала до цели
states <- matrix(c(start,dist(start)),1,2*N+4)

# рабочий список
work <- 1

men   <- function(s) s[1:N]           # мужчины
women <- function(s) s[(N+1):(2*N)] # женщины
boat  <- function(s) s[2*N+1]         # лодка
prev  <- function(s) s[2*N+2]         # предыдущее состояние
depth <- function(s) s[2*N+3]         # глубина

# проверка состояния на допустимость
is_available <- function(s)
{
  idx <- (1:N)[men(s)!=women(s)] # пары, где мужчина и женщина не рядом
```

```
  for (i in idx)
    if (any(men(s)[-i]==women(s)[i])) return(FALSE)

  TRUE
}
# проверка состояния на существование
exists <- function(s)
{
  for (i in 1:dim(states)[1])
    if (all(men(states[i,])==men(s)) &&
        all(women(states[i,])==women(s)) &&
        boat(states[i,])==boat(s)) return(TRUE)

  FALSE
}
# множество сторон, на которые можно переместиться из текущего состояния
move <- function(side) sides[sides!=side]

# обработка состояния-кандидата аналогична обработке при поиске в ширину
process <- function(s)
{
  if (is_available(s) && !exists(s))
  {
    states <<- rbind(states,c(s,dist(s)))
    work <<- c(work,dim(states)[1])
  }
}
# генерация потомков состояния с индексом idx
generate <- function(idx)
{
  s <- states[idx,]
  d <- depth(s)+1

  # один мужчина
  for (i in 1:N)
  {
    if (men(s)[i]!=boat(s)) next
    for (side in move(men(s)[i]))
    {
      m <- men(s)
      m[i] <- side

      process(c(m,women(s),side,idx,d))
    }
  }
  # двое мужчин
  for (i in 1:(N-1))
  for (j in (i+1):N)
  {
    if (men(s)[i]!=men(s)[j] || men(s)[i]!=boat(s)) next
    for (side in move(men(s)[i]))
    {
      m <- men(s)
```

```
      m[c(i,j)] <- side

      process(c(m,women(s),side,idx,d))
    }
  }
  # одна женщина
  for (i in 1:N)
  {
    if (women(s)[i]!=boat(s)) next
    for (side in move(women(s)[i]))
    {
      w <- women(s)
      w[i] <- side

      process(c(men(s),w,side,idx,d))
    }
  }
  # две женщины
  for (i in 1:(N-1))
  for (j in (i+1):N)
  {
    if (women(s)[i]!=women(s)[j] || women(s)[i]!=boat(s)) next
    for (side in move(women(s)[i]))
    {
      w <- women(s)
      w[c(i,j)] <- side

      process(c(men(s),w,side,idx,d))
    }
  }
  # мужчина и женщина
  for (i in 1:N)
  {
    if (men(s)[i]!=women(s)[i] || women(s)[i]!=boat(s)) next
    for (side in move(men(s)[i]))
    {
      m <- men(s)
      w <- women(s)

      m[i] <- side
      w[i] <- side

      process(c(m,w,side,idx,d))
    }
  }
}

while(TRUE)
{
  # если рабочий список пуст, то решения не существует
  if (length(work)==0) stop('решение не найдено...', call.=FALSE)
  # достигнуто целевое состояние
  if (all(states[work[1],1:(2*N)]==goal)) break
```

```
  idx <- work[1]
  work <- work[-1]

  generate(idx)

  # сортировка рабочих состояний в порядке возрастания расстояния
  o <- order(states[work,2*N+4])
  work <- work[o]
}

# формирование решения
solution <- work[1]
while (prev(states[solution[1],]) != -1)
  solution <- c(prev(states[solution[1],]),solution)

# печать решения
for (s in solution)
  cat(states[s,1:N],'␣-␣',
      states[s,(N+1):(2*N)],'␣-␣',
      states[s,2*N+1],'\n')
```

```
0 0 0 0  -  0 0 0 0  -  0
0 0 0 0  -  2 2 0 0  -  2
0 0 0 0  -  0 2 0 0  -  0
0 0 0 0  -  1 2 1 0  -  1
0 0 0 0  -  0 2 1 0  -  0
0 1 1 0  -  0 2 1 0  -  1
0 0 1 0  -  0 2 1 0  -  0
1 0 1 0  -  1 2 1 0  -  1
1 0 1 0  -  2 2 1 0  -  2
1 0 1 0  -  2 0 1 0  -  0
1 1 1 1  -  2 0 1 0  -  1
2 1 1 1  -  2 0 1 0  -  2
1 1 1 1  -  1 0 1 0  -  1
1 1 1 1  -  0 0 1 0  -  0
1 1 1 1  -  1 1 1 0  -  1
1 1 1 0  -  1 1 1 0  -  0
1 1 1 1  -  1 1 1 1  -  1
```

❑

## 3.3 Объектно-ориентированное программирование

### 3.3.1 Система S3

В S3 нет формального определения класса. Объект имеет «форму» класса, если он имеет атрибут-строку, который рассматриваемая как имя класса:

```
class(x)
class(x) <- value
unclass(x)
```

```
inherits(x, what, which = FALSE)
isa(x, what)
```

**Пример.** *Простая реализация интервальной арифметики.*

Если гарантируется, что $x \in [x_1, x_2]$ и $y \in [y_1, y_2]$, то

$$
\begin{aligned}
&x + y \in [x_1 + y_1, x_2 + y_2],\\
&x - y \in [x_1 - y_2, x_2 - y_1],\\
&x \times y \in [\min z, \max z], \quad z = \{x_1y_1, x_1y_2, x_2y_1, x_2y_2\},\\
&x/y \in x \times [1/y_2, 1/y_2], \quad [y_1, y_2] \not\ni 0.
\end{aligned}
$$

```
# as.interval(c()) — [0,0]
# as.interval(c(1)) или as.interval(1) — [1,1]
# as.interval(c(2,3)) — [2,3]
# as.interval(c(2,3,4,...)) — сообщение об ошибке (слишком большая длина аргумента)
# as.interval(x), где x — интервал, возвращает копию x
as.interval <- function(x)
{
  if (!inherits(x, 'interval'))
  {
    n = length(x)
    if (n == 0) res <- c(0,0)
    if (n == 1) res <- c(x,x)
    if (n == 2) res <- x
    if (n>2) stop("количество аргументов больше 2")

    class(res) <- c('interval', class(x))
  }
  else res <- x

  res
}
# проверка, является ли x интервалом
is.interval <- function(x) inherits(x, 'interval')

# печать интервала x= [x1,x2]
# если x1 = x2, то печатается x1, иначе — [x1,x2]
print.interval <- function(x)
{
  if (x[1]==x[2]) cat(x[1],"\n") else cat("[",x[1],x[2],"]\n")
}

# арифметические операции
'+.interval' <- function(x,y)
{
  x1 <- as.interval(x)
  x2 <- as.interval(y)
  as.interval(c(x1[1]+x2[1],x1[2]+x2[2]))
}
'-.interval' <- function(x,y)
{
  x1 <- as.interval(x)
```

```
  x2 <- as.interval(y)
  as.interval(c(x1[1]-x2[2],x1[2]-x2[1]))
}
'*.interval' <- function(x,y)
{
  x1 <- as.interval(x)
  x2 <- as.interval(y)
  t <- c(x1[1]*x2[1],x1[1]*x2[2],x1[2]*x2[1],x1[2]*x2[2])
  as.interval(c(min(t),max(t)))
}
'/.interval' <- function(x,y)
{
  x1 <- as.interval(x)
  x2 <- as.interval(y)
  if (x2[1]<=0 && 0<=x2[2]) stop("деление на ноль")
  as.interval(x1*c(1/x2[2],1/x2[1]))
}
```

❑

**Пример** (использование интервальной арифметики S3). *Оценка реальной процентной ставки:*

$$\mathrm{RIR} = \frac{C_0}{C_1}(\mathrm{NIR} - 1) \approx \mathrm{NIR} - \frac{C_1 - C_0}{C_0}.$$

*Здесь*

- RIR — *реальная процентная ставка (real investment rate);*
- NIR — *номинальная процентная ствка (nominal investment rate);*
- $C_0$ *и* $C_1$ — *индексы цен в начале и конце рассматриваемого временно́го периода.*

```
NIR <- 0.07
C0 <- as.interval(c(1.19,1.21))
C1 <- as.interval(c(1.22,1.26))

# RIR = C0/C1(1+NIR)-1
C0/C1*as.interval(1+NIR)-as.interval(1)
```

```
[ 0.01055556 0.06122951 ]
```

```
# RIR ≈ NIR - (C1-C0)/C0
as.interval(NIR)-(C1-C0)/C0
```

```
[ 0.01117647 0.06173554 ]
```

❑

### 3.3.2 Система S4

Для формального определения нового класса используется функция `setClass`:

```
setClass(Class, slots, contains=character())
```

Здесь

- `Class` — имя класса в виде строки;
- `slots` — поля класса;
- `contains` — классы, наследником которых является текущий класс.

**Пример.** *Простая реализация интервальной арифметики.*

```
Interval <- setClass(
  'Interval',
  slots = c(bounds = 'numeric'),
  prototype = list(bounds = c(0.0,0,0)),
  validity = function(object)
  {
    if (object@bounds[1] > object@bounds[2])
    {
      cat('Ошибка: нижняя граница больше верхней\n')
      return(FALSE)
    }
    return(TRUE)
  }
)
setGeneric(name='interval',
  def = function(x) { standardGeneric('interval') }
)
```

`Interval` — название класса. `interval` — имя функции, используемой для создания объекта `Interal`.

```
# x -> [x,x] и c(x1,x2) -> [x1,x2]
setMethod(f="interval",
  signature="numeric",
  definition = function(x)
  {
    n = length(x)
    if (n == 1) res <- Interval(bounds=c(x,x))
    else
      if (n == 2) res <- Interval(bounds=x)
      else stop("Неправильное количество аргументов")
    return(res)
  }
)
# [x1,x2] -> [x1,x2]
setMethod(f="interval",
  signature="Interval",
  definition = function(x) { return(x) }
)
# x -> [x,x] и c(x1,x2) -> [x1,x2]
```

```
setMethod(f='interval',
  signature='numeric',
  definition = function(x)
  {
    n = length(x)
    if (n == 1) res <- Interval(bounds=c(x,x))
    else
      if (n == 2) res <- Interval(bounds=x)
      else stop('Неправильное количество аргументов')
    return(res)
  }
)
```

$\# [x_1, x_2] \to [x_1, x_2]$

```
setMethod(f='interval',
  signature='Interval',
  definition = function(x) { return(x) }
)
```

Вывод на экран представления интервала (интервал нулевой ширины представляется одним числом):

```
setMethod('show','Interval',
  function(object)
  {
    if (object@bounds[1] == object@bounds[2]) cat(object@bounds[1],'\n')
    else cat('[',object@bounds[1],object@bounds[2],']\n')
  }
)
```

Определение арифметических операций над интервалами см. на стр. 96.

```
setMethod('+',c('Interval','Interval'),
  function(e1,e2) {
    bnd <- e1@bounds+e2@bounds
    interval(bnd)
  }
)
setMethod('-',c('Interval','Interval'),
  function(e1,e2)
  {
    lb <- e1@bounds[1]-e2@bounds[2]
    ub <- e1@bounds[2]-e2@bounds[1]
    interval(c(lb,ub))
  }
)
setMethod('*',c('Interval','Interval'),
  function(e1,e2)
  {
    t <- c()
    for (i in 1:2) for (j in 1:2) t <- c(t,e1@bounds[i]*e2@bounds[j])
    interval(c(min(t),max(t)))
  }
)
setMethod('/',c('Interval','Interval'),
  function(e1,e2)
  {
```

```
    if (e2@bounds[1]<=0 && 0<=e2@bounds[2]) stop('Деление на ноль')
    e1 * interval(c(1/e2@bounds[2],1/e2@bounds[1]))
  }
)
```

❑

**Пример** (использование интервальной арифметики S4)**.** *Оценка реальной процентной ставки (см. с. 97).*

```
NIR <- 0.07
C0 <- Interval(bounds=c(1.19,1.21)) # явное определение объекта
C1 <- interval(c(1.22,1.26))        # использование функции-генератора

# RIR = C0/C1(1 + NIR) – 1
C0/C1*interval(1+NIR)-interval(1)
```

```
[ 0.01055556 0.06122951 ]
```

```
# RIR ≈ NIR – (C1–C0)/C0
interval(NIR)-(C1-C0)/C0
```

```
[ 0.01117647 0.06173554 ]
```

❑

## 3.4 Функциональное программирование

При функциональном программировании (см., например, [1] и [20]) функции имеют свойства переменных: они могут быть аргументами других функций, а также возвращаться в качестве результата.

**Пример.** *Поиск стационарных точек гладкой функции* $f\colon \mathbb{R} \to \mathbb{R}$ *методом Ньютона-Рафсона, в котором методом Ньютона решается уравнение* $f'(x) = 0$*:*

$$x_0, \quad x_i = x_{i-1} - \frac{f'(x_{i-1})}{f''(x_{i-1})}, \quad i = 1, 2, \ldots$$

Для оценки значений первой и второй производной функции $f$ в точке $x$ можно использовать двухстороннюю аппроксимацию, дающую погрешность порядка $h^2$:

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2), \quad f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h^2).$$

Программной реализацией этих выражений будут функции высших порядков `diff` и `diff2`:

```
diff <- function(f,h=0.1) function(x) (f(x+h)-f(x-h))/(2*h)
diff2 <- function(f,h=0.1) function(x) (f(x+h)-2*f(x)+f(x-h))/h^2
```

При реализации метода Ньютона-Рафсона можно использовать итерацию (текущее состояние хранится в переменной `x`):

```
solver <- function(x0,f,eps=0.01)
{
  x <- x0
  newton <- function(x) x-diff(f)(x)/diff2(f)(x)
  repeat
  {
    x_next <- newton(x)
    if (abs(x-x_new)<eps) return (x_new)
    x <- x_next
  }
}
```

или рекурсию (без сохранения состояния):

```
solver <- function(x0,f,eps=0.01)
{
  newton <- function(x) x-diff(f)(x)/diff2(f)(x)
  newton_iter <- function(x)
  {
    x_next <- newton(x)
    if (abs(x-x_next)<eps) return(x_next)
    newton_iter(x_next)
  }
  newton_iter(x0)
}
```

❑

# Глава 4

# Графика

## 4.1 Двухмерные графики

$$y = f(x), \quad x_1 \leqslant x \leqslant x_2$$

### 4.1.1 Основные команды

Чаще всего используются команды

- `plot` — создание нового или обновление существующего графического окна с добавлением графика;
- `points` — добавление на существующий рисунок графика в виде набора точек;
- `lines` — добавление на существующий рисунок графика в виде ломаной.

Не так часто, но бывает полезна команда

- `text` — добавление текста на существующий рисунок.

#### Команда `plot`

Команда `plot` — обобщённая функция, создающая подходящий график, соответствующий типу отображаемого объекта `R`:

```
plot(x, ...)
plot(x, y, ...)
```

Полный вариант:

```
plot(x, y = NULL, type = "p",  xlim = NULL, ylim = NULL,
     log = "", main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
     ann = par("ann"), axes = TRUE, frame.plot = axes,
     panel.first = NULL, panel.last = NULL, asp = NA,
     xgap.axis = NA, ygap.axis = NA,
     ...)
```

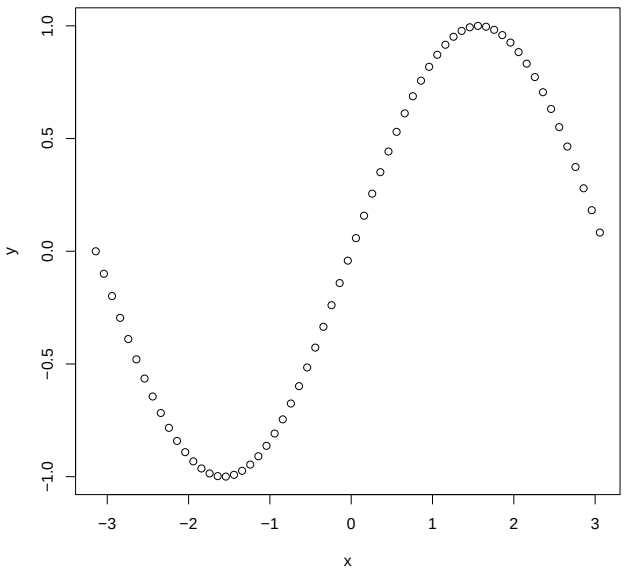
Необязательные параметры, имеющие значения по умолчанию:

- `type` — тип графика:
  - `p` — точки;

  - `l` — ломаная линия;
  - `b` — ломаная линия с разрывом в вершинах, где рисуются точки;
  - `c` — ломаная линия с разрывом в вершинах;
  - `o` — ломаная линия, в вершинах которой рисуются точки;
  - `h` — вертикальные линии;
  - `s` — кусочно-постоянная ломаная линия («ступеньки», stairs), высота ступеньки определяется значением в начале отрезка;
  - `p` — кусочно-постоянная ломаная линия («ступеньки», stairs), высота ступеньки определяется значением в конце отрезка;
  - `n` — ничего не рисуется (чистый холст с координатными осями и подписями);
- `main` — заголовок;
- `sub` — подзаголовок;
- `xlab` — название $x$-оси;
- `ylab` — название $y$-оси;
- `asp` — формат рисунка: отношение $y/x$.

По умолчанию просто отображаются точки в виде незакрашенных кружков с чёрным контуром:

```
x <- seq(from=-pi,to=pi,by=0.1)
y <- sin(x)
plot(x,y)
```




Можно нарисовать график заданной в виде процедуры функции $f(x)$, указав границы области определения аргумента $x_{\min} \leqslant x \leqslant x_{\max}$:

```
plot(f, xmin, xmax, ...)
```

Например, для функции Рунге

$$f(x) = \frac{1}{1+x^2}$$

```
f <- function(x) 1/(1+x^2)
```

на интервале $[-5, 5]$:

```
plot(f, -5,5, ylab='1/(1+x^2)')
```

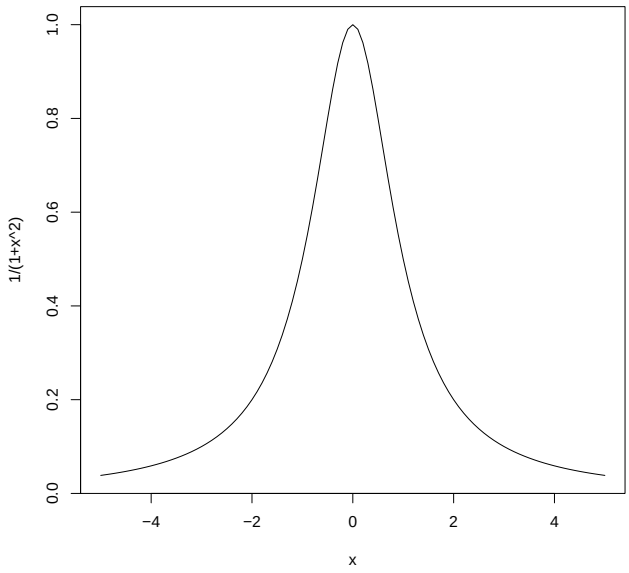
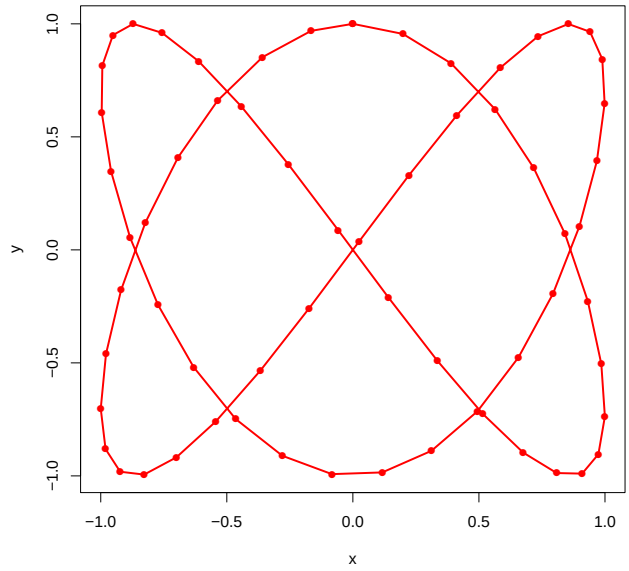



График функций $\big(x(t), y(t)\big)$, зависящих от параметра $t$ строится аналогично обычному графику зависимости $y$ от $x$:

```
t <- c(seq(from=0,to=2*pi,by=0.1),0)
x <- sin(2*t)
y <- cos(3*t)
plot(x,y,type='o',pch=16,lwd=2,col='red')
```




Если переменная $x$ представляет *качественную* величину, то необходимо сообщить об этом R, используя функцию `factor`.

**Пример** (графики качественных и количественных величин)**.** *Данные о зарплате (*W *гульденов в час, до вычета налогов) в Нидерландах в 1987 г. [12: с. 134]: 150 наблюдений по работающим на полную ставку не менее четырёх дней в неделю. Экзогенные переменные:* SEX *(пол),* EDU *(уровень образования) и* AGE *(возраст, лет).*

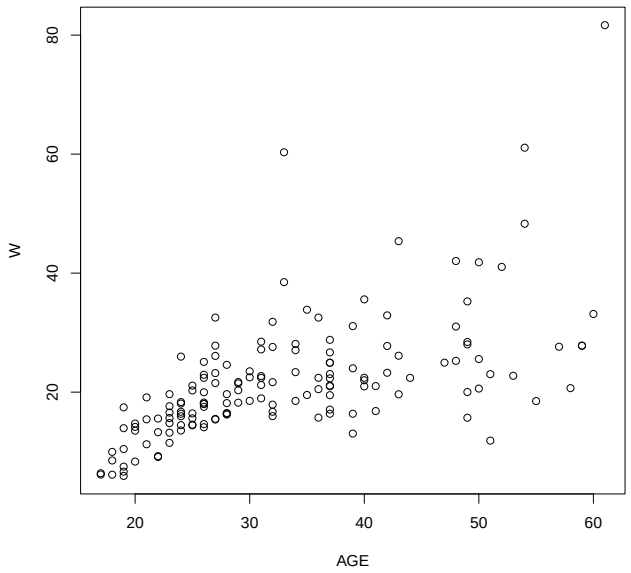
```
# Очистка рабочей памяти
rm(list=ls())
# Загружаем данные из файла
dat <- read.csv('d:/wages.csv',sep=';',dec=',')

# Преобразуем числа в качественные значения

# Пол: мужской, женский
dat$SEX <- factor(dat$SEX,labels=c('male','female'))
# Уровень образования: начальная школа или менее, низшее ремесленное,
# среднее, высшее ремесленное и университет
dat$EDU <- factor(dat$EDU,
          labels=c('low','lowprof','prof','highprof','univ'))
```

Команда `attach` добавляет данные в путь, по которому `R` выполняет поиск. После команды `attach` уже нет необходимости указывать название фрейма `dat`.

```
attach(dat)
```

График зависимости зарплаты от возраста:

```
plot(AGE,W)
```




Добавим цвета, чтобы различать мужчин и женщин:

```
points(AGE[SEX=='male'],W[SEX=='male'],pch=16,col='blue')
points(AGE[SEX=='female'],W[SEX=='female'],pch=16,col='red')
```

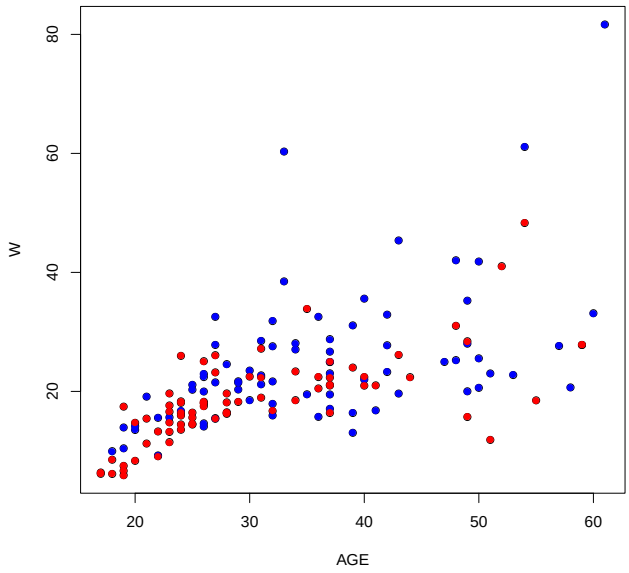



График зависимости зарплаты от пола (качественная величина):

```
plot(SEX,W,xlab='Sex',ylab='Wage',
     col=c(rgb(.8,.8,1),rgb(1,.8,.8)))
```

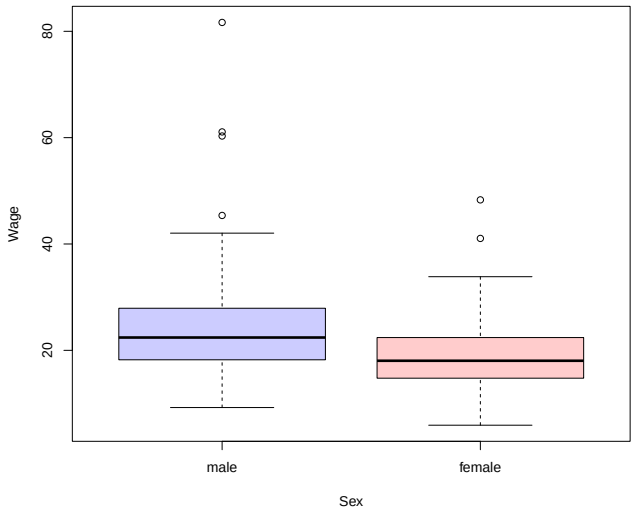
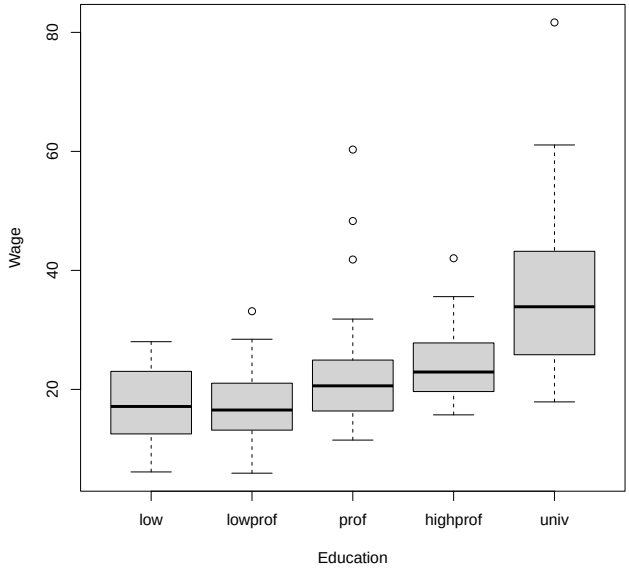


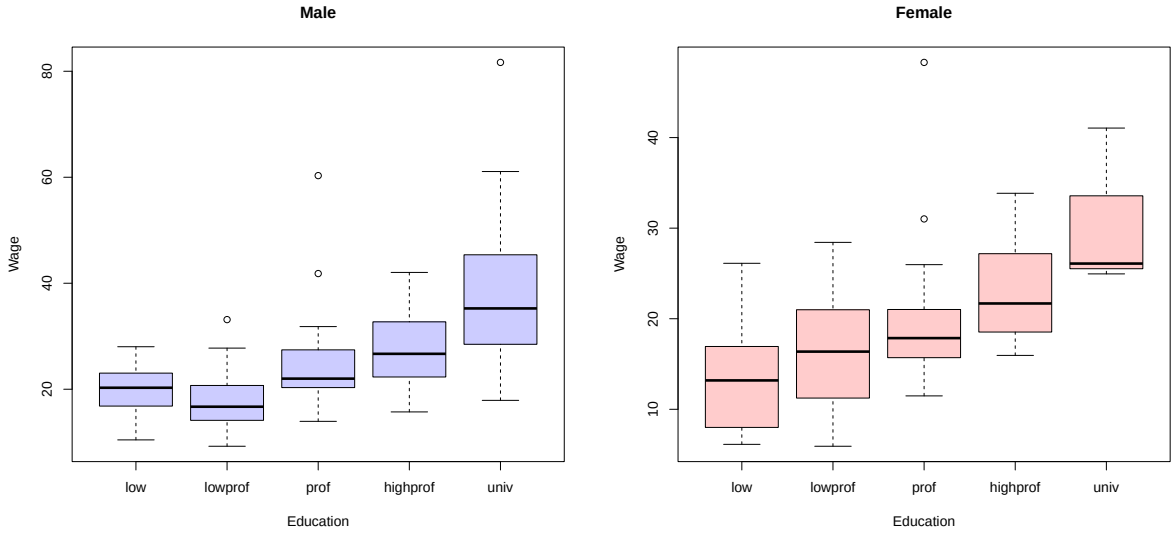

График зависимости зарплаты от уровня образования (тоже качественная величина):

```
plot(EDU,W,xlab='Education',ylab='Wage')
```




Больше информации можно получить, нарисовав графики зависимости зарплаты от уровня образования отдельно для мужчин и для женщин:

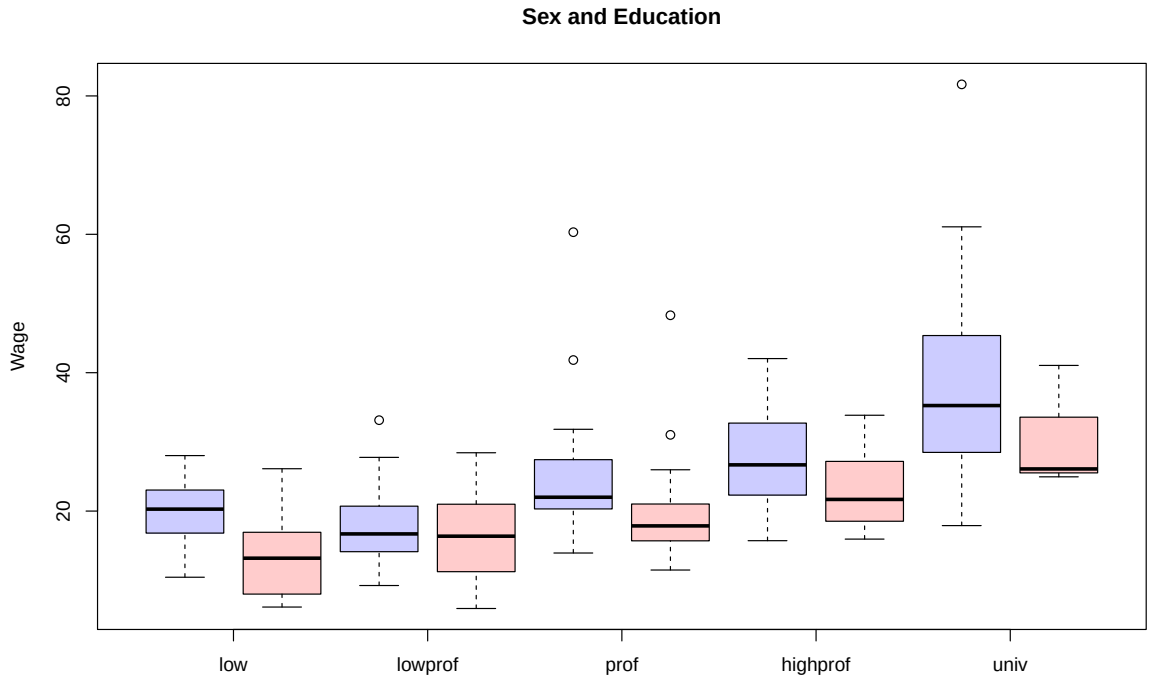
```
m <- SEX=='male'
plot(EDU[m],W[m],xlab='Education',ylab='Wage',main='Male',
     col=rgb(.8,.8,1))
plot(EDU[!m],W[!m],xlab='Education',ylab='Wage',main='Female',
     col=rgb(1,.8,.8))
```




Можно объединить графики зависимости зарплаты от уровня образования для мужчин и женщин на одном рисунке, используя цвет для определения пола:

```
# объединяем значения уровня образования и пола
SE <- c()
for (i in 1:dim(dat)[1])
 SE <- c(SE,paste(dat$EDU[i],dat$SEX[i]))
SE <- factor(SE)

# рисуем обединённый график без подписей у горизонтальной оси
```

```
plot(SE,W,main='Sex and Education',xlab='',ylab='Wage',
     col=rep(c(rgb(.8,.8,1),rgb(1,.8,.8)),5),xaxt='n')

# добавляем подписи-уровни образования
axis(1,at=seq(from=1.5,by=2,length=5),
       labels=c('low','lowprof','prof','highprof','univ'))
```




Команда `detach` удаляет данные из пути поиска.

```
detach(dat)
```

### Точки: команда `points`

Для добавления на существующий график точек используется команда `points`:

```
points(x, y = NULL, type = "p", ...)
```

Необязательный параметр `pch` позволяет указать вид точек, используемых на графике:

| | | | | |
|---|---|---|---|---|
| □ 0 | | | | |
| ○ 1 | △ 2 | + 3 | × 4 | ◇ 5 |
| ▽ 6 | ⊠ 7 | ✳ 8 | ⧆ 9 | ⊕ 10 |
| ⧈ 11 | ⊞ 12 | ⊠ 13 | ⧈ 14 | ■ 15 |
| ● 16 | ▲ 17 | ◆ 18 | ● 19 | • 20 |
| ○ 21 | □ 22 | ◇ 23 | △ 24 | ▽ 25 |

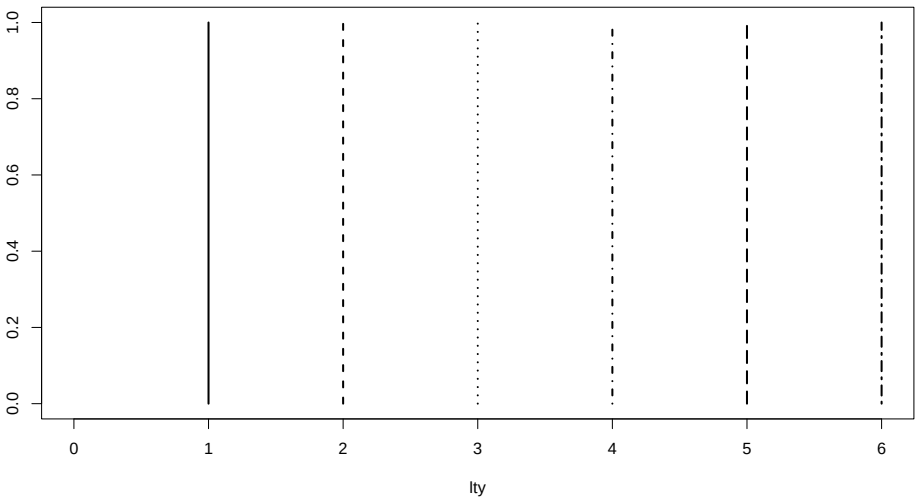
### Линии: команда `lines`

Для добавления на существующий график ломаной линии используется команда `lines`:

```
lines(x, y = NULL, type = "l", ...)
```

- `lwd` — толщина линии (по умолчанию 1).
- `lty` — тип линии (по умолчанию 1).
  - 0 — невидимая[1] (`blank`).
  - 1 — сплошная (`solid`).
  - 2 — штрихами `dashed`).
  - 3 — пунктиром (`dotted`).
  - 4 — штрихпунктирная (`dotdash`).
  - 5 — длинные штрихи (`longdash`).
  - 6 — штрихи и длинные штрихи (`twodash`).

```
plot(c(0,6),c(0,1),type='n',xlab='lty',ylab='')
for (i in 0:6) lines(c(i,i),c(0,1),lwd=2,lty=i)
```



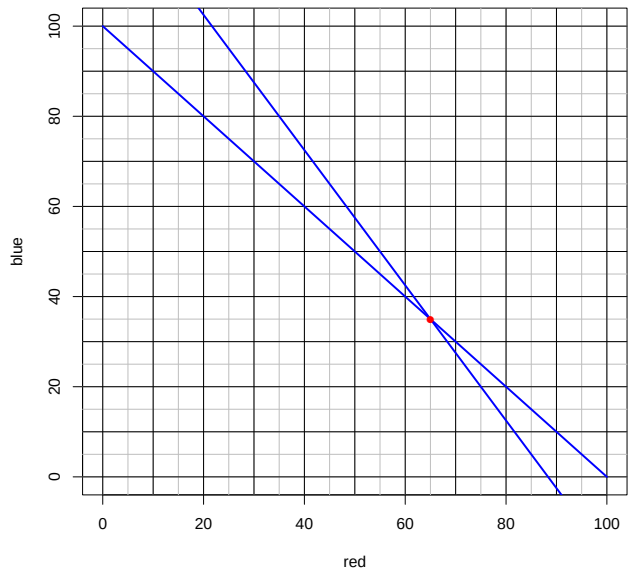

**Пример** (графическое решение системы из двух уравнений). *См. с. 32*

$$\textit{красная} + \textit{синяя} = 100$$
$$75 \cdot \textit{красная} + 50 \cdot \textit{синяя} = 6625$$

```
# исходные данные
red <- 0:100
blue1 <- 100-red
blue2 <- (6625-75*red)/50

# чистый холст для рисования
plot(c(0,100),c(0,100),
```

[1] Вы суслика видите? Нет? *А он есть!*
Пастафарианцы, используя тип линии 0, могут рисовать невидимых розовых единорогов.

```
     type='n',xlab='red',ylab='blue')

# сетка
for (z in seq(from=0,to=100,by=5))
{
  abline(h=z,col='grey')
  abline(v=z,col='grey')
}
for (z in seq(from=0,to=100,by=10))
{
  abline(h=z)
  abline(v=z)
}

# собственно графики, соответствующие уравнениям
lines(red,blue1,col='blue',lwd=2)
lines(red,blue2,col='blue',lwd=2)

# точка пересечения
points(65,35,pch=16,col='red')
```




❑

**Текст: команда** `text`

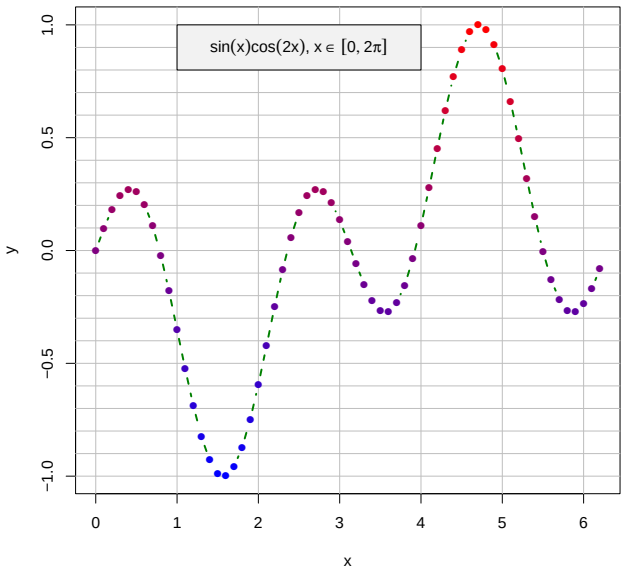
Для добавления на существующий график текста используется команда `text`:

```
text(x, y = NULL, labels = seq_along(x$x), adj = NULL,
     pos = NULL, offset = 0.5, vfont = NULL,
     cex = 1, col = NULL, font = NULL, ...)
```

**Пример.** *Добавление надписи на график функции*

$$\sin(x)\cos(2x), \quad 0 \leqslant x \leqslant 2\pi.$$

```
x <- seq(from=0,to=2*pi,by=0.1)
y <- sin(x)*cos(2*x)
cl <- (y-min(y))/(max(y)-min(y))

# холст
plot(x,y,type='n')

# фоновая решётка
for (i in seq(from=-1,to=1,by=0.1)) abline(h=i,col='grey')
for (i in 0:6) abline(v=i,col='grey')

# линия штрихами
lines(x,y,lwd=2,type='c',col=rgb(0,.5,0),lty='dashed')
# точки, цвет зависит от значения y
for (i in 1:length(x))
  points(x[i],y[i],pch=16,col=rgb(cl[i],0,1-cl[i]))

# фон текста
rect(1,.8,4,1,col=rgb(.95,.95,.95))

# текст
text(2.5,.9,expression(paste(sin(x)*cos(2*x),',␣',
                             x %in% group('[',list(0,2*pi),']'))))
```




❑

### Графические параметры: команда par

Команда par позволяет посмотреть текущие или задать новые значения графических параметров:

```
par(параметр=значение, ...)
```

```
par()
```

```
$xlog
[1] FALSE
$ylog
[1] FALSE
$adj
[1] 0.5
...
```

Команда `names` выводит список всех параметров, с которыми работает `par`:

```
names(par())
```

```
 [1] "xlog"      "ylog"      "adj"       "ann"       "ask"       "bg"
 [7] "bty"       "cex"       "cex.axis"  "cex.lab"   "cex.main"  "cex.sub"
[13] "cin"       "col"       "col.axis"  "col.lab"   "col.main"  "col.sub"
[19] "cra"       "crt"       "csi"       "cxy"       "din"       "err"
[25] "family"    "fg"        "fig"       "fin"       "font"      "font.axis"
[31] "font.lab"  "font.main"  "font.sub"  "lab"       "las"       "lend"
[37] "lheight"   "ljoin"     "lmitre"    "lty"       "lwd"       "mai"
[43] "mar"       "mex"       "mfcol"     "mfg"       "mfrow"     "mgp"
[49] "mkh"       "new"       "oma"       "omd"       "omi"       "page"
[55] "pch"       "pin"       "plt"       "ps"        "pty"       "smo"
[61] "srt"       "tck"       "tcl"       "usr"       "xaxp"      "xaxs"
[67] "xaxt"      "xpd"       "yaxp"      "yaxs"      "yaxt"      "ylbias"
```

### 4.1.2 Цвет

**Функция `rgb`**

Функция `rgb` позволяет определить цвет, используя аддитивную модель смешения красной, зелёной и синей компонент, а при необходимости также можно установить уровень прозрачности, задав значение альфа канала:

```
rgb(red, green, blue, alpha, names = NULL, maxColorValue = 1)
```

Здесь

- `red`, `green` и `blue` — вектора значения красной, зелёной и синей компонент в диапазоне $[0, m]$, где $m$ задаётся параметром `maxColorValue`.

  Если `maxColorValue = 255`, то для представления значений компонент используются целые числа из множества $\{0, 1, \ldots, 255\}$ и результат `rgb` вычисляется более эффективно.

- `alpha` — вектор значений альфа-канала, отвечающего за прозрачность. Значение в диапазоне $[0, m]$ (см. выше).

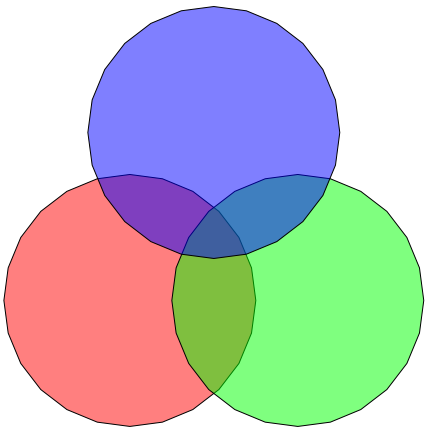
  При `alpha`= 0 цвет полностью прозрачен, при `alpha`= $m$ — полностью непрозрачен. Если 0 <`alpha`< $m$, то цвет точки смешивается с цветом фона.

- `names` — вектор названий получившихся цветов.

- `maxColorValue` — максимальное значение цветовых компонент и альфа-канала.

**Пример.** *Пересекающиеся полупрозрачные круги.*

```
disk <- function(x,y,r,n=25,...)
{
  t <- seq(from=0,to=2*pi,length=n)
  polygon(x+r*cos(t),y+r*sin(t),...)
}

plot(c(-2.5,2.5),c(-2.5,2.5),type='n',asp=1,xlab='',ylab='',axes=FALSE)
disk(-1,-1,1.5,col=rgb(1,0,0,.5))
disk(1,-1,1.5,col=rgb(0,1,0,.5))
disk(0,1,1.5,col=rgb(0,0,1,.5))
```



❑

**Пример.** *Цветовой треугольник.*

```
P1 <- c(-1,0)      # красная вершина
P2 <- c(1,0)       # зелёная вершина
P3 <- c(0,3^.5)    # синяя вершина

C1 <- c(1,0,0)
C2 <- c(0,1,0)
C3 <- c(0,0,1)

plot(c(-1,1),c(0,3^.5),type='n',xlab='',ylab='',axes=FALSE)

N <- 250
for (h in seq(from=0,to=1,length=N))
{
  x1 <- P1[1]+h*(P2[1]-P1[1]) # выпуклая комбинация красного и зелёного
  y1 <- P1[2]+h*(P2[2]-P1[2])
  c1 <- C1*(1-h)+C2*h

  x2 <- P1[1]+h*(P3[1]-P1[1]) # выпуклая комбинация красного и синего
  y2 <- P1[2]+h*(P3[2]-P1[2])
  c2 <- C1*(1-h)+C3*h

  for (d in seq(from=0,to=1,length=N))
```

```
  {
    x <- x1+d*(x2-x1) # координаты и цвета точки в треугольнике
    y <- y1+d*(y2-y1)
    C <- c1+d*(c2-c1)

    C <- C/max(C)      # нормализация цвета

    points(x,y,pch=20,col=rgb(C[1],C[2],C[3]))
  }
}
```

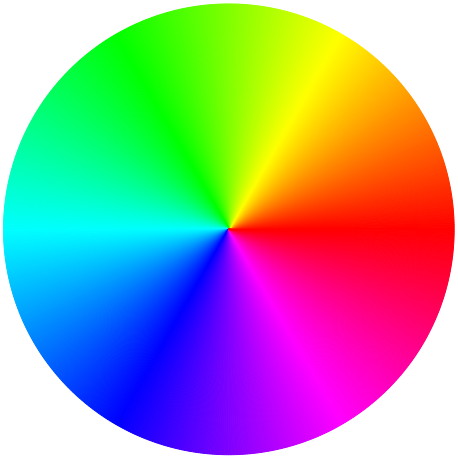
Координаты и цвет каждой точки треугольника являются выпуклой комбинацией координат и цветов его вершин.

❑

**Пример.** *Цветовой круг, составленный из сегментов, соответствующих красному, жёлтому, зелёному, аквамариновому, синему и фиолетовому цветам.*

```
sector <- function(x,y,r,a,...)
  polygon(c(x,x+r*cos(a[1]),x+r*cos(a[2]),x),
          c(y,y+r*sin(a[1]),y+r*sin(a[2]),y),...)

COLOR <- rbind(c(1,0,0), # красный
               c(1,1,0), # жёлтый
               c(0,1,0), # зелёный
               c(0,1,1), # аквамариновый
               c(0,0,1), # синий
               c(1,0,1), # фиолетовый
               c(1,0,0)) #

numcol <- dim(COLOR)[1]-1  # количество цветов
angle <- 2*pi/numcol       # угол сектора для пары цветов
n <- 100
h  <- 1/n                  # количество оттенков в секторе
ha <- angle/n
```

```
plot(c(-1,1),c(-1,1),type='n',xlab='',ylab='',axes=FALSE,asp=1)
for (i in 1:numcol)
for (j in 0:(n-1))
{
  cl <- COLOR[i,]+h*j*(COLOR[i+1,]-COLOR[i,]) # RGB-компоненты
  cl <- rgb(cl[1],cl[2],cl[3])                # цвет в 16-ричном формате
  sector(0,0,1,c(angle*(i-1)+j*ha,angle*(i-1)+(j+1)*ha),
         col=cl,border=cl)
}
```



❑

**Функция** `hsv`

Функция `hsv` позволяет определить цвет, используя оттенок, насыщенность и яркость, а при необходимости также можно установить уровень прозрачности, задав значение альфа канала:

```
hsv(h = 1, s = 1, v = 1, alpha)
```

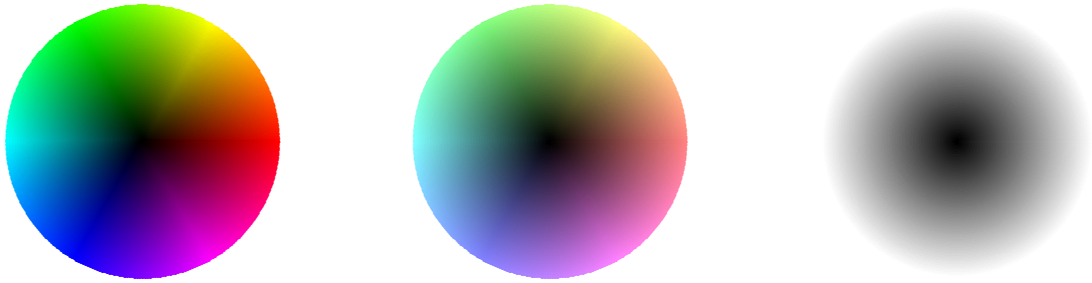
Здесь

- `h`, `s` и `v` — числовые векторы со значениями из интервала $[0,1]$ для оттенка (hue), насыщенности (saturation) и яркости (value);
- `alpha` — вектор значений альфа-канала (прозрачность);

  `alpha = 0` — полностью прозрачный, `alpha = 1` — непрозрачный.

```
# цветовой круг
plot(c(-1,1),c(-1,1),type='n',xlab='',ylab='',axes=FALSE,asp=1)

H <- seq(from=0,to=1,by=0.003)
V <- seq(from=0,to=1,by=0.02)
for (h in H)
for (v in V) points(v*cos(h*2*pi),v*sin(h*2*pi),pch=16,col=hsv(h,1,v))
```

Цветовые круги при насыщенности 1, 0.5 и 0:



### Функция hcl

Функция hcl позволяет определить цвет, используя оттенок, интенсивность и освещённость, а при необходимости также можно установить уровень прозрачности, задав значение альфа канала:

```
hcl(h = 0, c = 35, l = 85, alpha, fixup = TRUE)
```

- h — значение оттенка в диапазоне $[0, 360]$ (0 — красный, 120 — зелёный, 240 — синий).
- c — значение интенсивности в диапазоне $[0, 100]$. Максимальное значение зависит от h и l.
- l — значение освещённости в диапазоне $[0, 100]$. При фиксированных значениях h и c доступно только подмножество $[0, 100]$.
- alpha — значение альфа-канала (прозрачность) из интервала $[0, 1]$ (0 — полная прозрачность, 1 — полная непрозрачность).
- fixup — использовать коррекцию цвета при приведении к модели RGB.

```
# цветовой круг
plot(c(-1,1),c(-1,1),type='n',xlab='',ylab='',axes=FALSE,asp=1)
for (h in seq(from=0,to=1,by=0.003))
  for (cr in seq(from=0,to=1,by=0.02))
    points(cr*cos(h*2*pi),cr*sin(h*2*pi),lwd=2,col=hcl(h*360,cr*100,100))
```

Цветовые круги при освещённости 33, 66 и 100:

### Функция gray

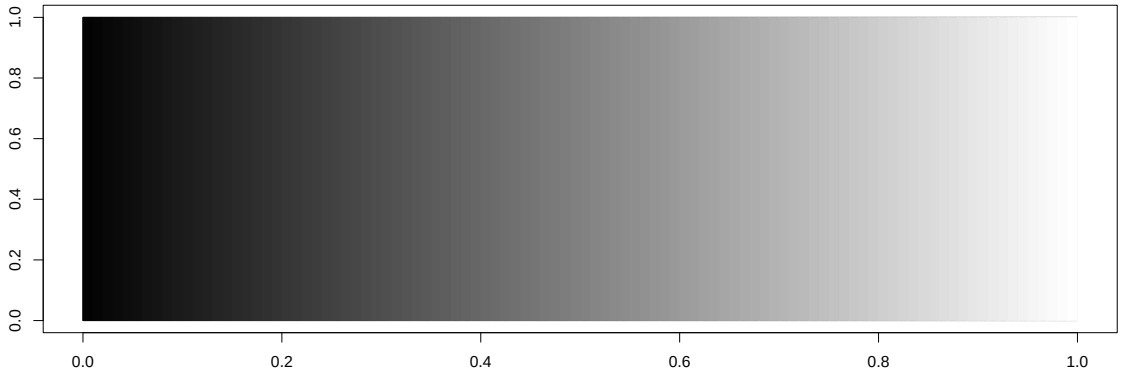
Функция gray используется при рисовании в оттенках серого:

```
gray(level, alpha)
grey(level, alpha)
```

Здесь

- `level` — вектор значений уровней серого цвета в диапазоне $[0, 1]$ ($0$ — чёрный цвет, $1$ — белый.
- `alpha` — вектор значений альфа-канала (прозрачность).

```
n <- 100
plot(c(0,1),c(0,1),type='n',xlab='',ylab='')
for (i in 1:n) rect((i-1)/n,0,i/n,1,col=gray(i/n),border=gray(i/n))
```




### Функции, основанные на палитрах

Следующие функции используют предопределённые палитры цветов:

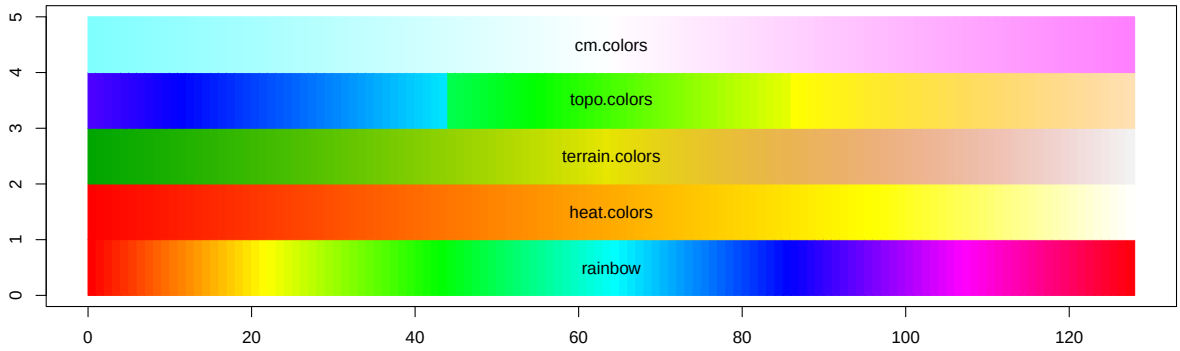
```
rainbow(n, s = 1, v = 1, start = 0, end = max(1, n - 1)/n,
        alpha, rev = FALSE)
heat.colors(n, alpha, rev = FALSE)
terrain.colors(n, alpha, rev = FALSE)
topo.colors(n, alpha, rev = FALSE)
cm.colors(n, alpha, rev = FALSE)
```

Здесь

- `n` — количество цветов в палитре.
- `rev` — цвета в обратном порядке.

```
n <- 128
plot(c(0,n),c(0,5),type='n',xlab='',ylab='')
for (i in 1:n)
{
  rect(i-1,0,i,1,col=rainbow(n)[i],border=rainbow(n)[i])
```

```
  rect(i-1,1,i,2,col=heat.colors(n)[i],border=heat.colors(n)[i])
  rect(i-1,2,i,3,col=terrain.colors(n)[i],border=terrain.colors(n)[i])
  rect(i-1,3,i,4,col=topo.colors(n)[i],border=topo.colors(n)[i])
  rect(i-1,4,i,5,col=cm.colors(n)[i],border=cm.colors(n)[i])
}
text(n/2,.5,'rainbow')
text(n/2,1.5,'heat.colors')
text(n/2,2.5,'terrain.colors')
text(n/2,3.5,'topo.colors')
text(n/2,4.5,'cm.colors')
```




Ещё предопределённые палитры:

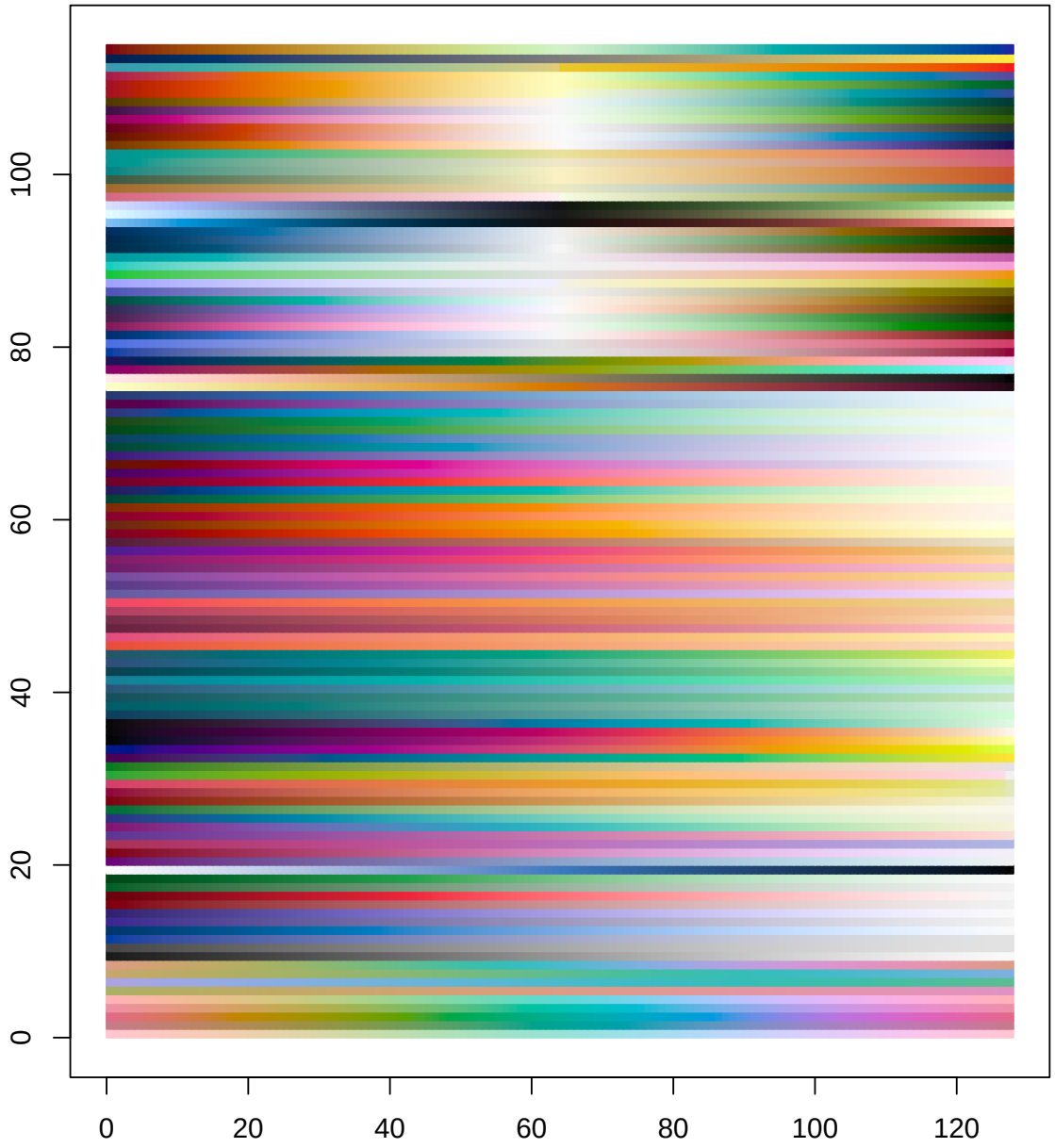
```
hcl.colors(n, palette = "viridis", alpha = NULL, rev = FALSE, fixup = TRUE)
hcl.pals(type = NULL)
```

- `n` — количество цветов в палитре.
- `palette` — название палитры.

```
hcl.pals()
```

```
  [1] "Pastel 1"      "Dark 2"        "Dark 3"        "Set 2"
  [5] "Set 3"         "Warm"          "Cold"          "Harmonic"
  [9] "Dynamic"       "Grays"         "Light Grays"   "Blues 2"
.............................................................
[109] "BrBG"          "RdYlBu"        "RdYlGn"        "Spectral"
[113] "Zissou 1"      "Cividis"       "Roma"
```

```
n <- 128
m <- length(hcl.pals())
plot(c(0,n),c(0,m),type='n',xlab='',ylab='')
for (j in 1:m)
{
  colors <- hcl.colors(n,palette=hcl.pals()[j])
  for (i in 1:n)
  {
    rect(i-1,j-1,i,j,col=colors[i],border=colors[i])
  }
}
```



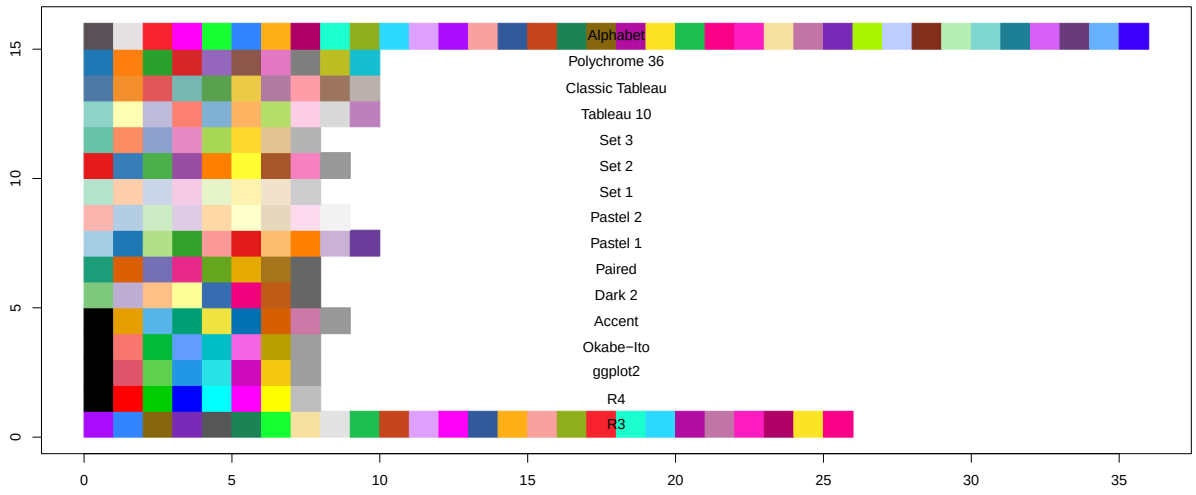

Палитры, использующие индексированные цвета.

```
palette(value)
palette.pals()
palette.colors(n = NULL, palette = "Okabe-Ito", alpha, recycle = FALSE)
```

```
palette.pals()
```

```
 [1] "R3"              "R4"              "ggplot2"         "Okabe-Ito"
 [5] "Accent"          "Dark 2"          "Paired"          "Pastel 1"
 [9] "Pastel 2"        "Set 1"           "Set 2"           "Set 3"
[13] "Tableau 10"      "Classic Tableau" "Polychrome 36"   "Alphabet"
```

```
pals <- palette.pals()
n <- max(sapply(pals,function(p) length(palette(p))))
m <- length(pals)
plot(c(0,n),c(0,m),type='n',xlab='',ylab='')
for (j in 1:m)
{
  colors <- palette(pals[j])
  for (i in 1:length(colors))
  {
    rect(i-1,j-1,i,j,col=colors[i],border=colors[i])
  }
  text(n/2,j-.5,palette.pals()[j])
}
```




## 4.2 Трёхмерные графики

Для рисования поверхности функции

$$y = f(x_1, x_2)$$

сначала следует подготовить исходные данные:

- построим одномерные сетки по координатам $x_1$ и $x_2$:

```
x1 <- seq(from=-5,to=5,by=0.25); n1 <- length(x1)
x2 <- seq(from=-5,to=5,by=0.25); n2 <- length(x2)
```

- определим функцию:

```
f <- function(x1,x2) x1^2-x2^2
```

- вычислим значения функции в узлах двумерной сетки, используя функцию `outer` (декартово произведение множеств):

```
y <- outer(x1,x2,f)
```

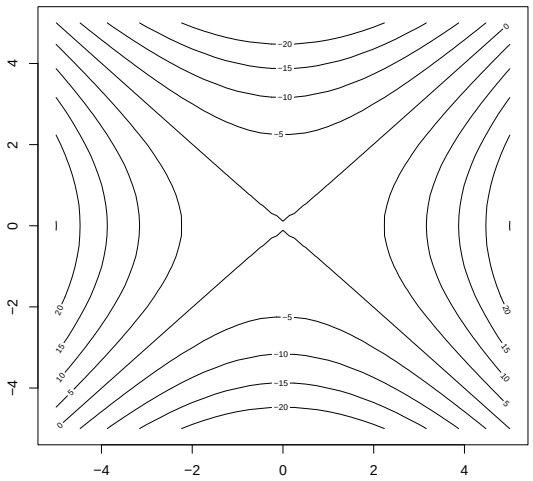
**График линий уровня** может быть простой контурный (функция countour)

```
contour(x = seq(0, 1, length.out = nrow(z)),
        y = seq(0, 1, length.out = ncol(z)),
        z,
        nlevels = 10, levels = pretty(zlim, nlevels),
        labels = NULL,
        xlim = range(x, finite = TRUE),
        ylim = range(y, finite = TRUE),
        zlim = range(z, finite = TRUE),
        labcex = 0.6, drawlabels = TRUE, method = "flattest",
        vfont, axes = TRUE, frame.plot = axes,
        col = par("fg"), lty = par("lty"), lwd = par("lwd"),
        add = FALSE, ...)
```

или с закрашенными областями (filled.contour):

```
filled.contour(x = seq(0, 1, length.out = nrow(z)),
               y = seq(0, 1, length.out = ncol(z)),
               z,
               xlim = range(x, finite = TRUE),
               ylim = range(y, finite = TRUE),
               zlim = range(z, finite = TRUE),
               levels = pretty(zlim, nlevels), nlevels = 20,
               color.palette = function(n) hcl.colors(n, "YlOrRd", rev = TRUE),
               col = color.palette(length(levels) - 1),
               plot.title, plot.axes, key.title, key.axes,
               asp = NA, xaxs = "i", yaxs = "i", las = 1,
               axes = TRUE, frame.plot = axes, ...)

.filled.contour(x, y, z, levels, col)
```

```
contour(x1,x2,y)
filled.contour(x1,x2,y)
```



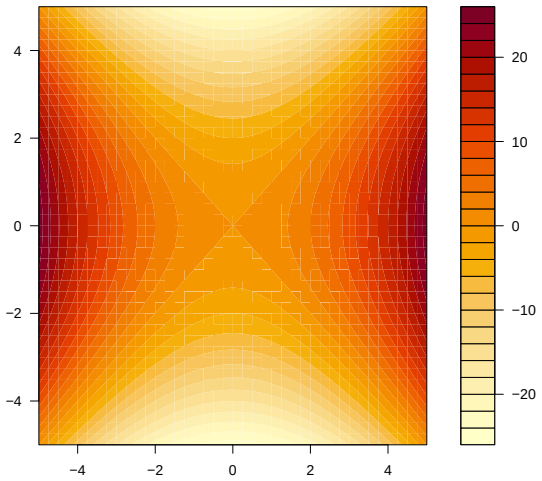
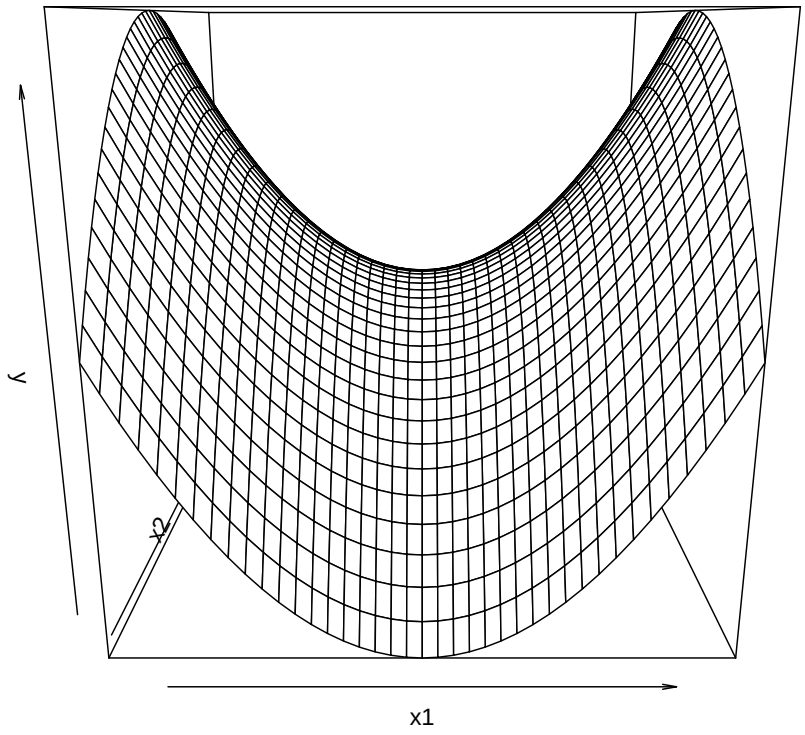



**График поверхности** строится функцией persp:

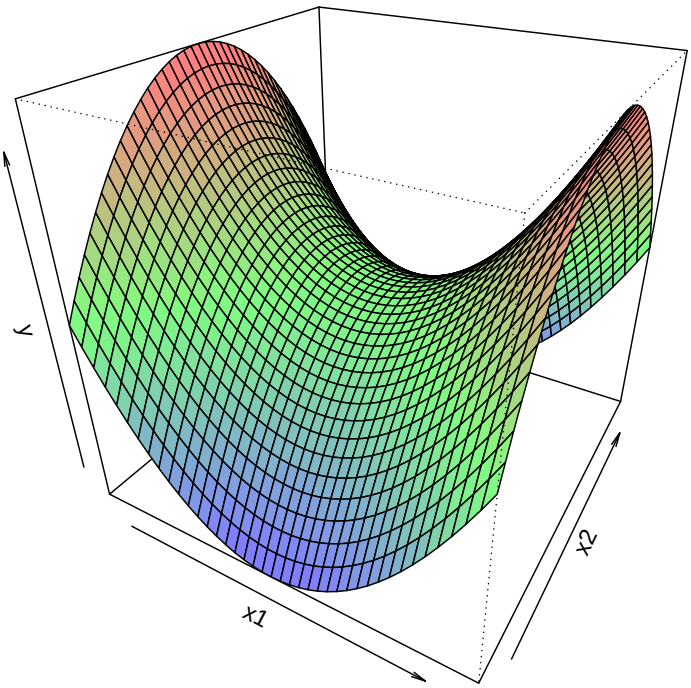
```
persp(x = seq(0, 1, length.out = nrow(z)),
      y = seq(0, 1, length.out = ncol(z)),
      z, xlim = range(x), ylim = range(y),
      zlim = range(z, na.rm = TRUE),
      xlab = NULL, ylab = NULL, zlab = NULL,
      main = NULL, sub = NULL,
      theta = 0, phi = 15, r = sqrt(3), d = 1,
      scale = TRUE, expand = 1,
      col = "white", border = NULL, ltheta = -135, lphi = 0,
      shade = NA, box = TRUE, axes = TRUE, nticks = 5,
      ticktype = "simple", ...)
```

```
persp(x1,x2,y)
```




Можно добавить на график цвет, зависящий от значения $y$-координаты точки (функция cut используется для преобразования количественных значений в качественные: числовой диапазон разбивается на поддиапазоны и элементу назначается качественное значение в зависимости от того, в какой диапазон он попал):

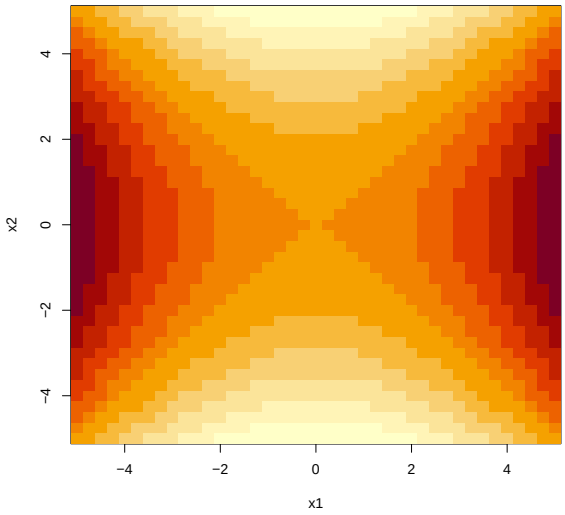
```
nc <- 32
colors <- colorRampPalette(c(rgb(.5,.5,1),
                             rgb(.5,1,.5),
                             rgb(1,.5,.5)))(nc)
ycol <- cut(y[-1,-1]+y[-1,-n2]+y[-n1,-1]+y[-n1,-n2],nc)

persp(x1,x2,y,col=colors[ycol],phi=30,theta=30)
```




Вместо линий уровня с помощью функции image можно нарисовать сетку, состоящей из раскрашенных прямоугольников, цвет которых зависит от величины z (а также от выбранной палитры):

```
image(x, y, z, zlim, xlim, ylim,
      col = hcl.colors(12, "YlOrRd", rev = TRUE),
      add = FALSE, xaxs = "i", yaxs = "i", xlab, ylab,
      breaks, oldstyle = FALSE, useRaster, ...)
```

```
image(x1,x2,y)
```




## 4.3 Рисование с нуля

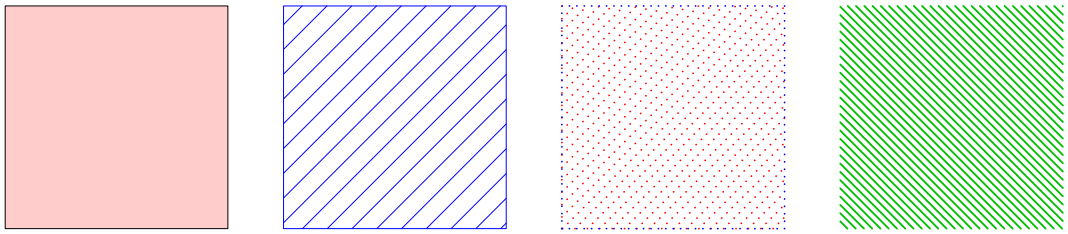
При рисовании можно использовать не только `points`, `lines` и `text`, но и команды, позволяющие создавать более сложные графические объекты из примитивов (прямоугольников, отрезков и т.п.).

### Прямоугольники

```
rect(xleft, ybottom, xright, ytop, density = NULL, angle = 45,
     col = NA, border = NULL, lty = par('lty'), lwd = par('lwd'),
     ...)
```

- `xleft`, `xright`, `ybottom`, `ytop` — координаты левой и правой, нижней и верхней границ прямоугольника. Векторы или скалярные величины.
- `density` — плотность штриховки (линий на дюйм). `NA` — штриховка отсутствует, позволяет выполнить сплошную заливку.
- `angle` — наклон линий штриховки.
- `col` и `border` — цвета штриховки (или заливки) и границы. `NA` означает отсутствие заливки (полная прозрачность, если не указано `density`) или отсутствие границы.
- `lty` — тип линий штриховки и границы. По умолчанию — сплошной (`solid`).
- `lwd` — толщина линий штриховки и границы.
- `...` — дополнительные графические параметры.

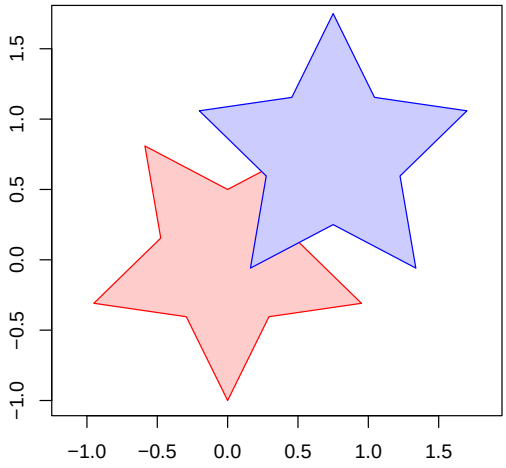
```
plot(c(0,20),c(0,4),type='n',xlab='',ylab='',axes=FALSE,asp=1)

rect(0,0,4,4,col=rgb(1,.8,.8))
rect(5,0,9,4,density=5,col='blue')
rect(10,0,14,4,density=10,col='red',border='blue',lty='dotted',lwd=2)
rect(15,0,19,4,density=15,angle=-45,col=rgb(0,.75,0),border=NA,lwd=2)
```



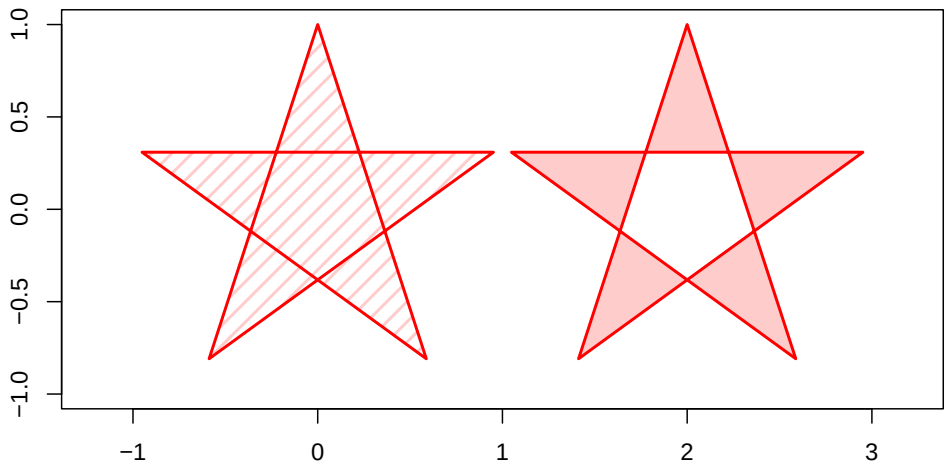
### Многоугольники

```
polygon(x, y = NULL, density = NULL, angle = 45,
        border = NULL, col = NA, lty = par('lty'),
        ..., fillOddEven = FALSE)
```

```
star <- function(cx,cy,radius,n,
                 scale=0.5,angle=0,canvas=NA,axes=FALSE,...)
{
  p <- complex(modulus=rep(c(radius,radius*scale),n),
               argument=seq(from=pi/2+angle,by=pi/n,length=2*n))
  p <- p + complex(real=cx,imaginary=cy)

  ind <- c(1:(2*n),1)

  if (!any(is.na(canvas)))
    plot(canvas[1:2],canvas[3:4],
         type='n',xlab='',ylab='',axes=axes,asp=1)

  polygon(Re(p[ind]),Im(p[ind]),...)
}

star(0,0,1,5,canvas=c(-1,1.7,-1,1.7),
     angle=pi/n,axes=TRUE,col=rgb(1,.8,.8),border='red')
star(.75,.75,1,5,
     col=rgb(.8,.8,1),border='blue')
```




```
n <- 5
p <- complex(modulus=rep(1,n),
             argument=seq(from=pi/2,by=2*pi/n,length=n))
ind <- c(1,3,5,2,4,1)
x <- Re(p[ind]); y <- Im(p[ind])

plot(c(-1,3),c(-1,1),type='n',xlab='',ylab='',asp=1)

polygon(x,y,col=rgb(1,.8,.8),border='red',lwd=2,density=10)
polygon(x+2,y,col=rgb(1,.8,.8),border='red',lwd=2)
```

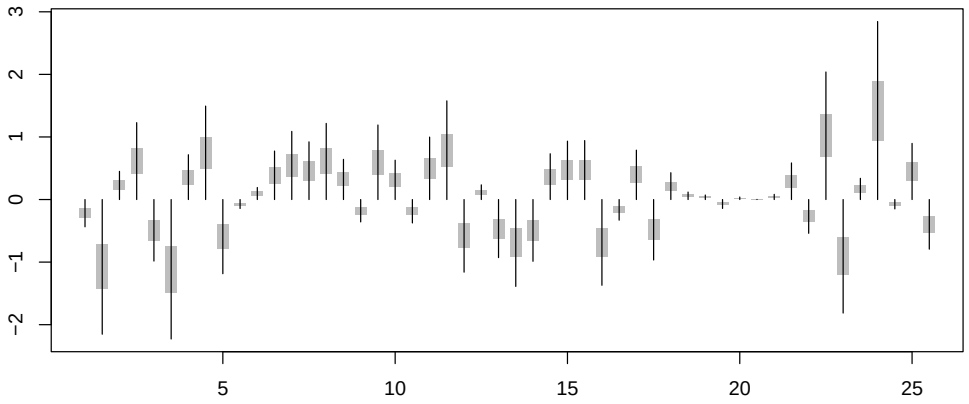



**Отрезки** между парами точек.

```
segments(x0, y0, x1 = x0, y1 = y0,
         col = par('fg'), lty = par('lty'), lwd = par('lwd'),
         ...)
```

- `x0`, `y0` — координаты начала.
- `x1`, `y1` — координаты конца.

```
n <- 50; h <- .5
x <- seq(from=1,by=h,length=n)
y <- rnorm(n)

plot(c(min(x),max(x)),c(min(y,0),max(y,0)),type='n',xlab=NA,ylab=NA)
for (i in 1:n)
     rect(x[i]-h/3,y[i]/3,x[i]+h/3,y[i]*2/3,border=NA,col='gray')

segments(x,rep(0,n),x,y)
```




**Стрелки** между парами точек.

```
arrows(x0, y0, x1 = x0, y1 = y0, length = 0.25, angle = 30,
       code = 2, col = par('fg'), lty = par('lty'),
       lwd = par('lwd'), ...)
```

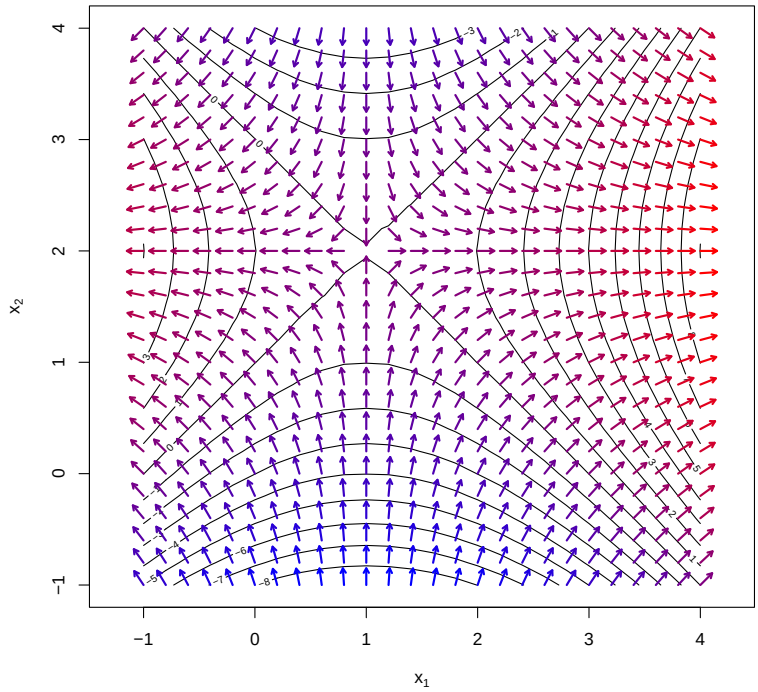
- `x0`, `y0` — координаты начала.
- `x1`, `y1` — координаты конца.
- `length` — длина линий наконечника стрелки.
- `angle` — угол между линией наконечника и её древком.
- `code` — код *типа* стрелки:
  1. начало ← конец;
  2. начало → конец;
  3. начало ↔ конец.

**Пример.** *Поле направлений векторов градиентов функции*

$$f(x_1, x_2) = (x_1 - 1)^2 - (x_2 - 2)^2$$

*на графике линий уровня.*

```
f <- function(x) (x[1]-1)^2-(x[2]-2)^2
df <- function(x) 2*c(x[1]-1,-x[2]+2)

T <- seq(from=-1,to=4,by=0.2)
n <- length(T)
y <- array(dim=c(n,n))
for (i in 1:n) for (j in 1:n) y[i,j] <- f(c(T[i],T[j]))

plot(c(min(T),max(T)),c(min(T),max(T)),type='n',
     xlab=expression(x[1]),ylab=expression(x[2]),asp=1)
contour(T,T,y,nlevels=15,
        xlab=expression(x[1]),ylab=expression(x[2]),asp=1)

y <- y-min(y)
y <- y/max(y)
for (i in 1:n) for (j in 1:n)
{
  ar <- df(c(T[i],T[j]))
  ar <- ar/sum(ar^2)^0.5 *0.15

  arrows(T[i],T[j],T[i]+ar[1],T[j]+ar[2],length=.05,
         col=rgb(y[i,j],0,1-y[i,j]),lwd=2)
}
```




### Прямые линии

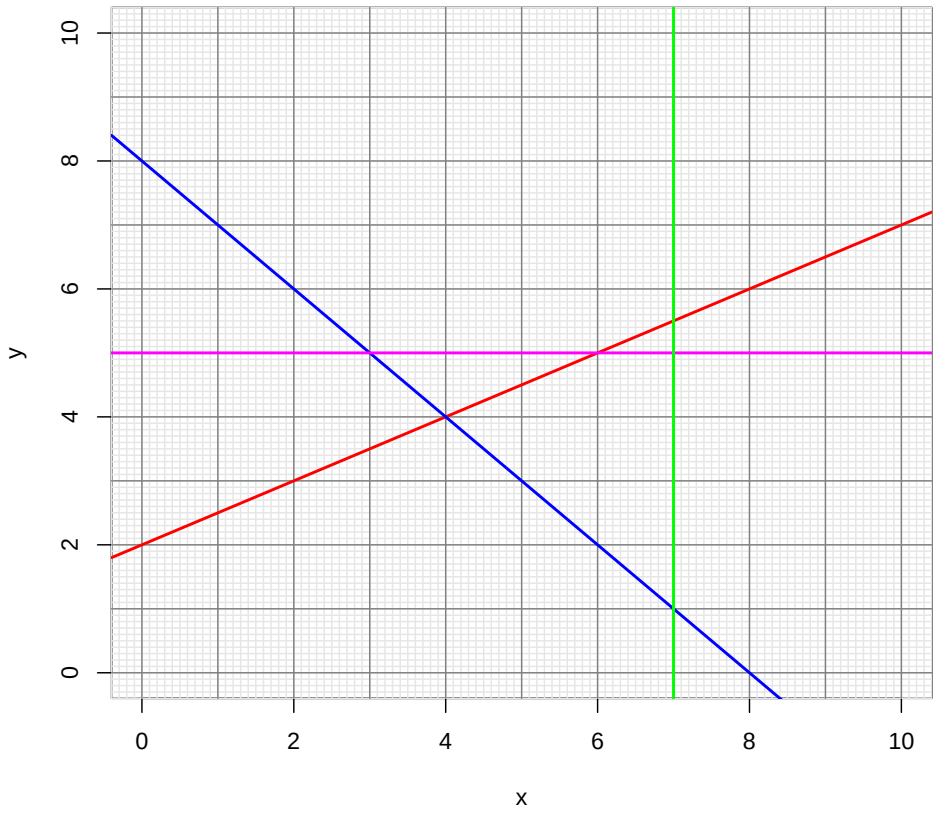
```
abline(a = NULL, b = NULL, h = NULL, v = NULL, reg = NULL,
       coef = NULL, untf = FALSE, ...)
```

- `a` и `b` — коэффициенты линейной функции $y = a + bx$.
- `coef` — двухэлементный вектор коэффициентов линейной функции

$$y = coeff_1 + coeff_2 \times x.$$

- `h` — $y$-значение *горизонтальной* линии.
- `v` — $x$-значение *вертикальной* линии.

```
# холст
plot(c(0,10),c(0,10),type='n',xlab='x',ylab='y')
# мультисетка
for (i in seq(from=-1,to=11,by=0.1))
{
  abline(h=i,col=rgb(.9,.9,.9))
  abline(v=i,col=rgb(.9,.9,.9))
}
for (i in seq(from=0,to=10))
{
  abline(h=i,col=rgb(.5,.5,.5))
  abline(v=i,col=rgb(.5,.5,.5))
}
# линии
abline(2,.5,lwd=2,col='red')                # y = 2 + 0.5x
abline(coef=c(8,-1),lwd=2,col='blue')       # y = 8 - x
abline(h=5,lwd=2,col='magenta')             # y = 5
abline(v=7,lwd=2,col='green')               # x = 7
```




## 4.4 Дополнительно

### 4.4.1 Библиотека ggplot2

Более подробное описание работы с `ggplot2` см. в [29].

### Функция `qplot`

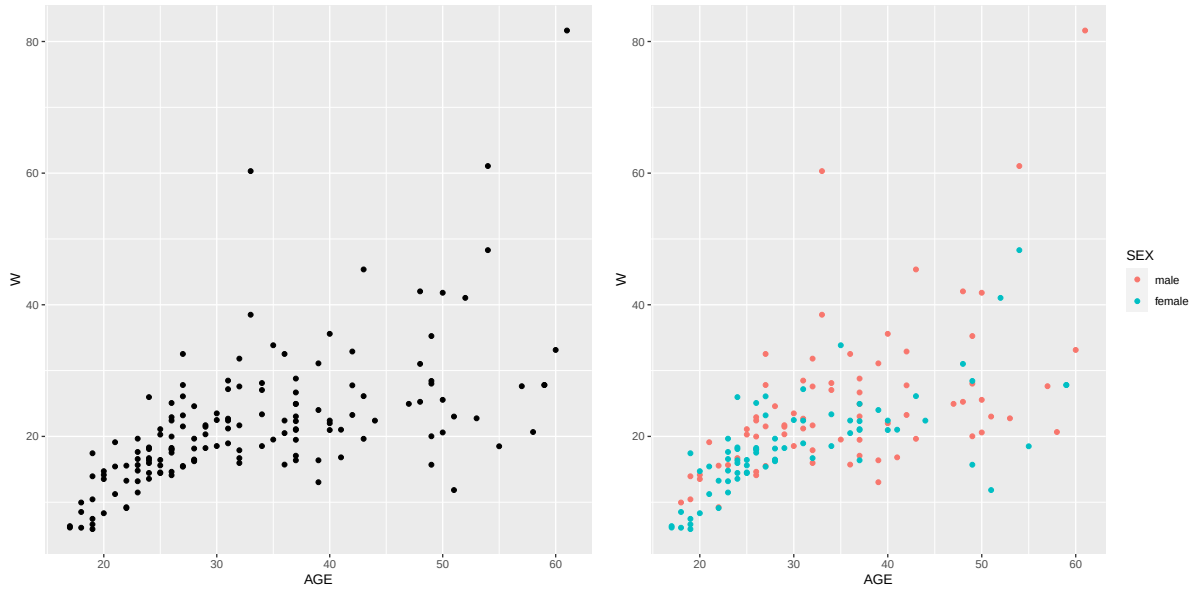
Функция `qplot` из библиотеки `ggplot2` предназначена для упрощения построения сложных графиков и разработана очень похожей на обычную функцию `plot`.

**Пример.** *Данные о зарплате (`W` гульденов в час, до вычета налогов) в Нидерландах в 1987 г. [12: с. 134]: 150 наблюдений по работающим на полную ставку не менее четырёх дней в неделю. Экзогенные переменные:* `SEX` *(пол),* `EDU` *(уровень образования) и* `AGE` *(возраст, лет).*

```
# Загружаем данные из файла
dat <- read.csv('d:/wages.csv',sep=';',dec=',')

# Преобразуем числа в качественные значения

# Пол: мужской, женский
dat$SEX <- factor(dat$SEX,labels=c('male','female'))
# Уровень образования: начальная школа или менее, низшее ремесленное,
# среднее, высшее ремесленное и университет
dat$EDU <- factor(dat$EDU,
          labels=c('low','lowprof','prof','highprof','univ'))
```

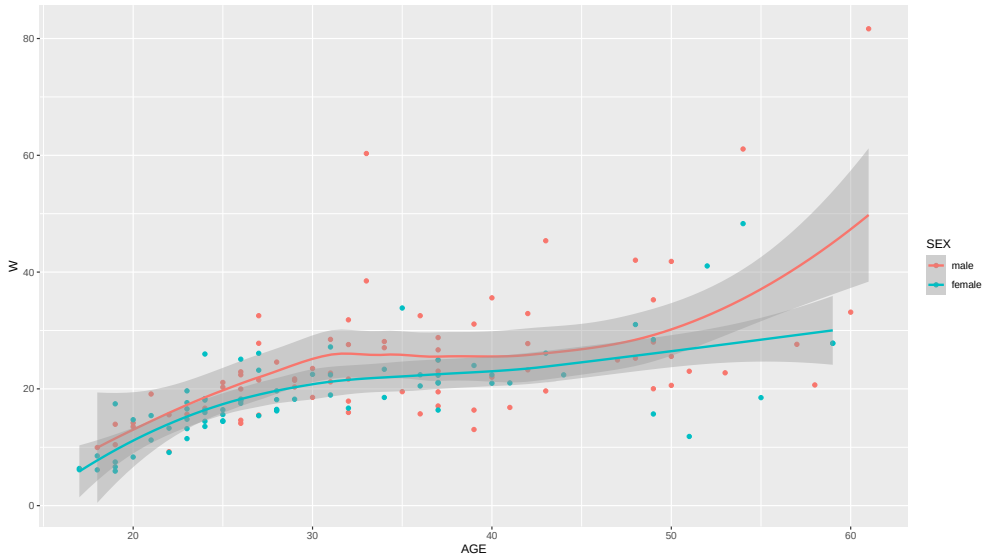
Зависимость зарплаты от возраста.

```
library(ggplot2)
qplot(AGE,W,data=dat)
qplot(AGE,W,data=dat,color=SEX)
```

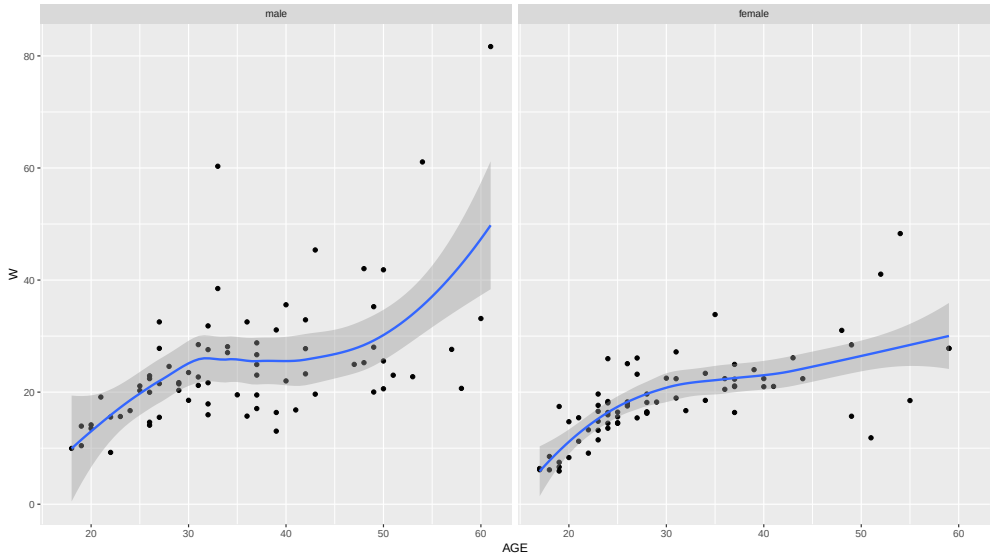



Зависимость зарплаты от возраста плюс сглаживание:

```
qplot(AGE,W,data=dat,color=SEX)+geom_smooth()
```

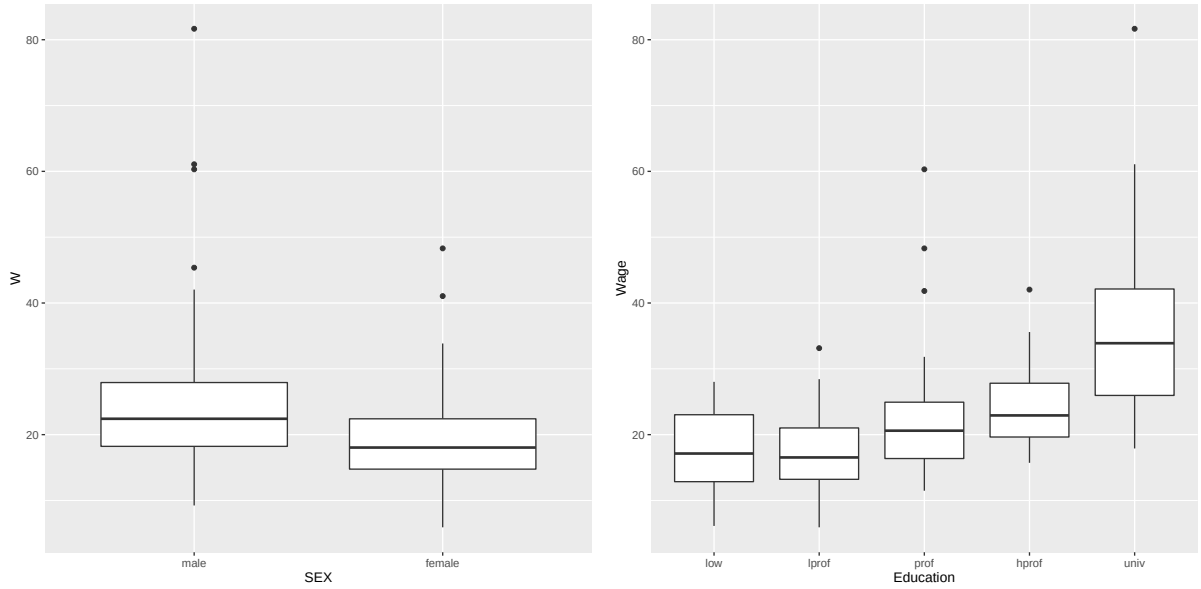



```
qplot(AGE,W,data=dat,facets=.~SEX)+geom_smooth()
```



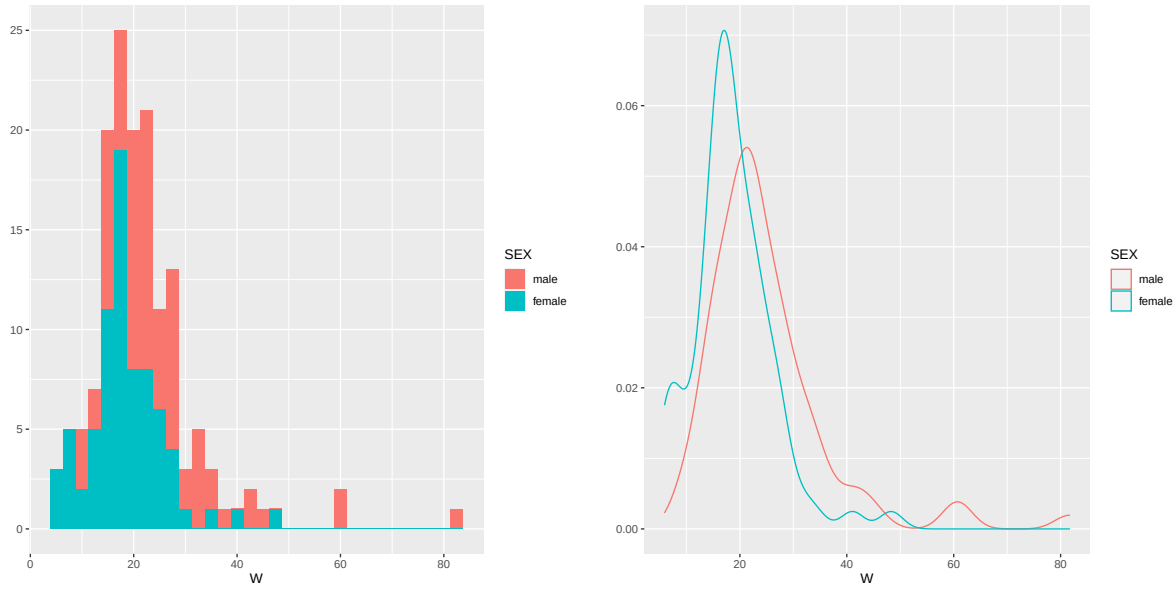

Зарплата и пол. Зарплата и уровень образования.

```
qplot(SEX,W,data=dat,geom='boxplot')
qplot(EDU,W,data=dat,geom='boxplot',xlab='Education',ylab='Wage')
```




Зарплата и пол: гистограмма и график плотности вероятности.

```
qplot(W,data=dat,geom='histogram',fill=SEX,binwidth=2.5)
qplot(W,data=dat,geom='density',color=SEX)
```




### 4.4.2 Библиотека `rgl`

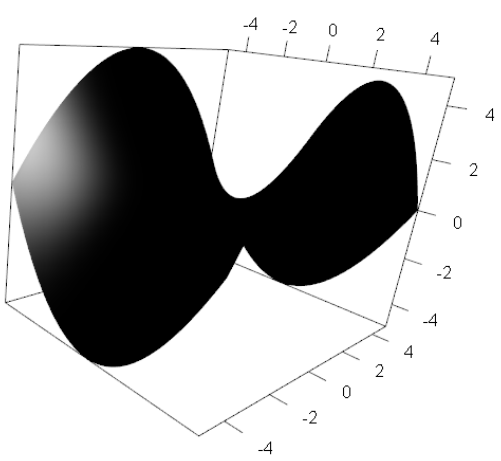
Библиотека `rgl` существенно использует OpenGL, подробное описание концепций которого можно найти, например, в [21].

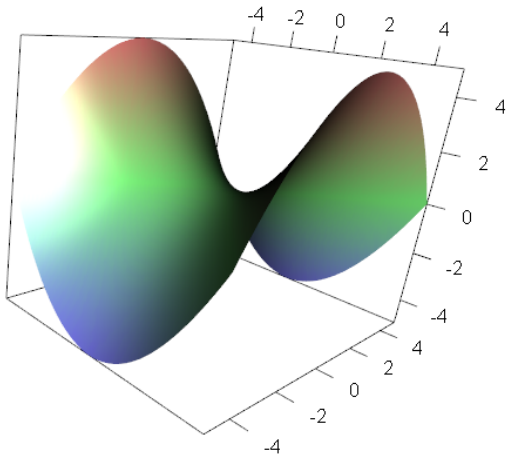
**Пример.** *Седловая поверхность*

$$z(x, y) = x^2 - y^2.$$

```
library(rgl)
open3d()
```

```
x <- seq(from=-5,to=5,by=0.1); n1 <- length(x)
y <- seq(from=-5,to=5,by=0.1); n2 <- length(y)

z <- outer(x,y,function(x,y) x^2-y^2)
# нормирование значений
z <- (z-min(z))/(max(z)-min(z))*10-5

# чёрно-белая поверхность
surface3d(x,y,z)

# цвет зависит от высоты (z-координаты)
zlim <- range(z)
zlen <- (zlim[2]-zlim[1])*5+1

colors <- colorRampPalette(c(rgb(.5,.5,1),
                             rgb(.5,1,.5),
                             rgb(1,.5,.5)))(zlen)
zcol <- colors[(z-zlim[1])*5+1]

surface3d(x,y,z,color=zcol,meshcolor='facets')
axes3d()

# текущая матрица преобразование
par3d()$userMatrix
# задание новой точки зрения
mt <- array(c( .75,-.25, .65,  0,
               .7, .15, -.7, 0,
              .15,  .9,  .3,  0,
                0,   0,   0,  1),dim=c(4,4))
view3d(userMatrix = mt)

# запись изображения в файл (только в формате png)
rgl.snapshot('d:/rgl.png', fmt='png', top=TRUE)
```



### 4.4.3 Библиотека `plotly`

**Пример** (Странный аттрактор Лоренца)**.** *Решить задачу Коши*

$$\begin{aligned} x'(t) &= -sx(t) + sy(t), & x(0) &= -8, \\ y'(t) &= -x(t)z(t) + rx(t) - y(t), & y(0) &= 8, \\ z'(t) &= x(t)y(t) - bz(t), & z(0) &= 27, \end{aligned}$$

*где* $b = 8/3$, $s = 10$, $r = 28$. *Нарисовать график решения в фазовом пространстве* $(x, y, z)$, *а также графики зависимости* $x$, $y$ *и* $z$ *от* $t$.

Правая часть системы дифференциальных уравнений:

```
lorenz <- function(t,x)
{
  b <- 8/3
  s <- 10
  r <- 28

  c(-s*x[1]+s*x[2],
    -x[1]*x[3]+r*x[1]-x[2],
    x[1]*x[2]-b*x[3])
}
```

Для численного решения системы дифференциальных уравнений будем использовать метод Рунге-Кутты 4-5 порядка точности:

```
rk45 <- function(t,x,f,h)
{
  k1 <- f(t,x)
  k2 <- f(t+h/2,x+h/2*k1)
  k3 <- f(t+h/2,x+h/2*k2)
  k4 <- f(t+h,x+h*k3)
  (k1+2*k2+2*k3+k4)/6
}
```

Исходные данные и собственно решение:

```
n <- 3000
h <- 0.01
t <- double(n+1);          t[1] <- 0
x <- array(dim=c(3,n+1)); x[,1] <- c(-8,8,27)

for (i in 1:n)
{
  x[,i+1] <- x[,i] + h*rk45(t[i],x[,i],lorenz,h)
  t[i+1]  <- t[i] + h
}
```

Оформление результатов в виде фрейма данных и трёхмерный график (функция `plot_ly`):

```
data <- data.frame(time=t,x=x[1,],y=x[2,],z=x[3,])

plot_ly(data, x = ~x, y = ~y, z = ~z,
        type = 'scatter3d', mode = 'lines',
        line = list(width=2,color='blue'))
```

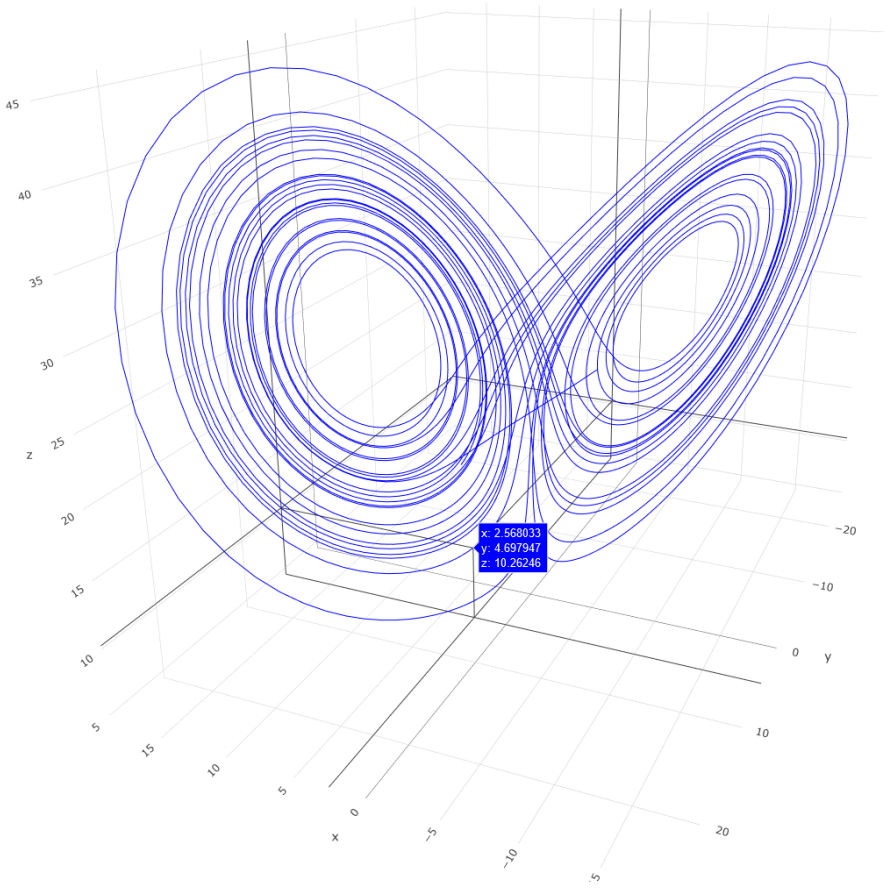
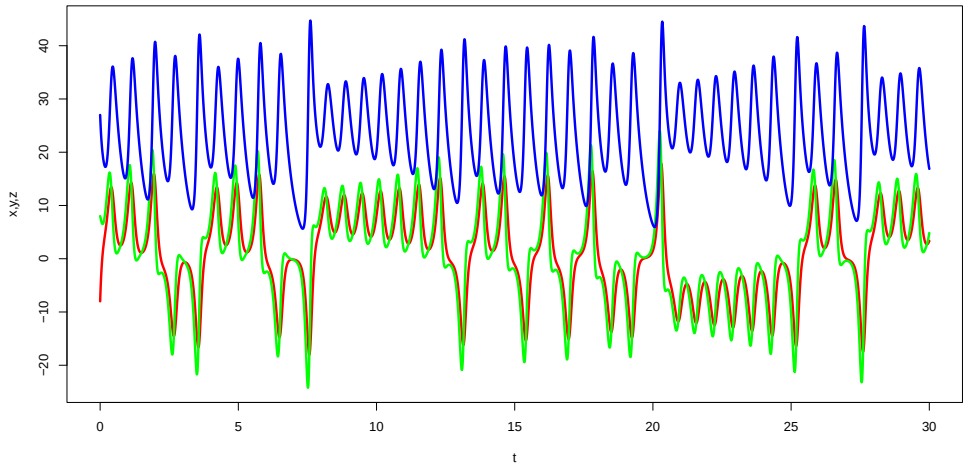



График зависимости значений $x$, $y$ и $z$ от $t$:

```
plot(t,x[1,],col='red',lwd=3,ylab='x,y,z',
     type='l',ylim=c(min(x),max(x)))
lines(t,x[2,],col='green',lwd=3)
lines(t,x[3,],col='blue',lwd=3)
```



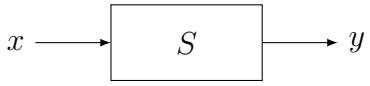

Если используется библиотека `ggplot2`, то функция `ggplotly` позволяет из статичных графиков получить динамические, интерактивные изображения.

# Глава 5

# Эконометрика

## 5.1 Модель множественной линейной регрессии

Любая экономическая система $S$ связана с окружающей средой. Эта связь заключается как во влиянии среды на систему (вход $x$), так и влиянии системы на среду (отклик $y$):




Сигнал $x$, который формируется вне системы, называется *экзогенной* (внешней) переменной. Значение отклика $y$ полностью определяется внутри системы, поэтому $y$ называется *эндогенной* (внутренней) переменной. Если внутренняя структура системы $S$ не известна (или мы не обращаем на неё внимания при построении модели), то мы имеем дело с моделью непрозрачного «чёрного ящика».

Модель множественной линейной регрессии основана на предположении, что поведение этого «чёрного ящика» можно описать в виде линейной зависимости значения эндогенной переменной $y$ от значений экзогенных переменных $x$:

$$y = \underbrace{\beta_0 + \sum_{j=1}^{m} \beta_j x_j}_{\hat{y}} + \varepsilon \tag{5.1}$$

Здесь

- $x$ — входные значения (регрессоры, факторы, независимые переменные, экзогенные переменные);
- $y$ — выходные значения (отклик, зависимые переменные, эндогенные переменные);
- $\hat{y}$ — модельные значения;
- $\beta$ — параметры;
- $\varepsilon$ — погрешность (шум, ошибка модели).

Линейная зависимость $y$ от $x$ означает, что изменение значения переменной $x_j$ на единицу, вне зависимости от исходного значения $x$, всегда влечёт изменение модельного значения $\hat{y}$ на $\beta_j$. Другими словами, скорость изменения $\hat{y}$ при изменении $x_j$ всегда постоянна и не зависит от значения $x_j$:

$$\frac{\partial \hat{y}}{\partial x_j} = \beta_j.$$

Предположим, что имеется некоторый набор данных $(x_i, y_i)$, $i = 1, \ldots, n$, для описания которого (по некоторым соображениям) подходит линейная модель. Подставив в (5.1) вместо $y$ и $x_j$ конкретные значения, получим систему уравнений

$$\begin{aligned} y_1 &= \beta_0 + \beta_1 x_{1,1} + \beta_2 x_{1,2} + \cdots + \beta_m x_{1,m} + \varepsilon_1, \\ y_2 &= \beta_0 + \beta_1 x_{2,1} + \beta_2 x_{2,2} + \cdots + \beta_m x_{2,m} + \varepsilon_2, \\ &\vdots \\ y_n &= \beta_0 + \beta_1 x_{n,1} + \beta_2 x_{n,2} + \cdots + \beta_m x_{n,m} + \varepsilon_n, \end{aligned}$$

которую можно переписать в векторно-матричной форме

$$y = X\beta + \varepsilon,$$

где

$$y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad X = \begin{pmatrix} 1 & x_{1,1} & x_{1,2} & \ldots & x_{1,m} \\ 1 & x_{2,1} & x_{2,2} & \ldots & x_{2,m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n,1} & x_{n,2} & \ldots & x_{n,m} \end{pmatrix}, \quad \beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_m \end{pmatrix}, \quad \varepsilon = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix}.$$

**Пример** (Линейная модель)**.**

Предположим, что истинная зависимость имеет вид

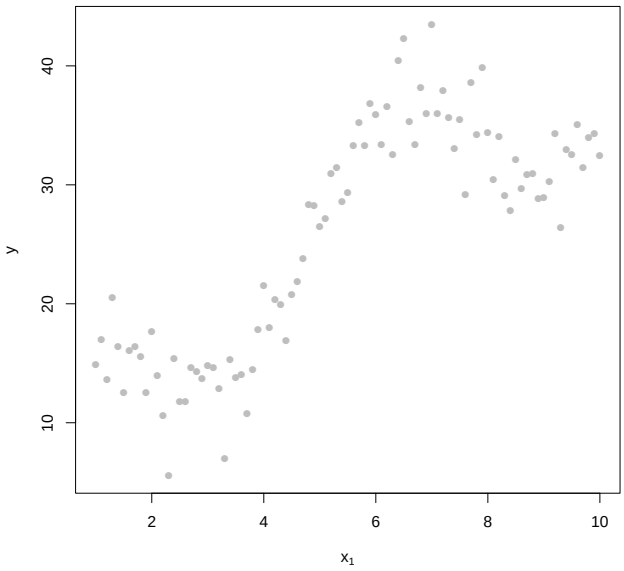
$$y = \underbrace{15 + 3x_1 + 7x_2}_{\hat{y}} + \varepsilon,$$

где $x_1 = 1, 1.1, 1.2, \ldots, 10$, $x_2 = \cos(x_1)$ и $\varepsilon \sim \mathcal{N}(0, 3^2)$. Тогда

```
set.seed(1)              # если требуется повторяемость результатов

x1 <- seq(from=1,to=10,by=0.1)
x2 <- cos(x1)
n  <- length(x1)         # объём выборки
m <- 2                   # число факторов в модели
eps <- rnorm(n,sd=3)
y <- 10 + 3*x1 + 7*x2 + eps
```

График исходных данных $(x_1, y)$:

```
plot(x1,y,pch=16,col='gray',
     xlab=expression(x[1]),ylab='y')
```




### 5.1.1 Метод наименьших квадратов

Теоретическая погрешность модели имеет вид $\varepsilon = y - \hat{y}$, и чем ближе значения $\varepsilon$ к нулю, тем более точной является модель. В методе наименьших квадратов подбираются такие значения параметров модели, чтобы

$$\sum_{i=1}^{n} \varepsilon_i^2 \to \min .$$

**Использование функции `R optim`**

```
X <- cbind(1,x1,x2)

ESS <- function(b) sum((y-X%*%b)^2)
optim(1:dim(X)[2], ESS)


$par
[1] 10.484576  2.976973  7.349443
...
```

По умолчанию функция `optim` применяет модифицированный Нелдером и Мидом метод поиска по симплексу (см. [15: стр. 87–93]). Альтернативой является использование более быстрого и точного квазиньютоновского метода Бройдена-Флетчета-Гольдфарба-Шанно (см. [15: стр. 128]):

```
optim(1:dim(X)[2], ESS,method='BFGS')


$par
[1] 10.485161  2.976840  7.349443
...
```

### Явное решение системы нормальных уравнений

Для вывода системы нормальных уравнений в матричной форме

1. применим к исходной модели левую трансформацию Гаусса:

$$y = X\beta + \varepsilon \quad \Rightarrow \quad X^\top y = X^\top X\beta + X^\top \varepsilon;$$

2. и выберем такие значения коэффициентов $\beta$, чтобы второе слагаемое, $X^\top\varepsilon$, обратилось в ноль:

$$X^\top y = X^\top X \underbrace{\beta}_{\to b} + \underbrace{X^\top \varepsilon}_{\to 0} \quad \Rightarrow \quad X^\top y = X^\top X b.$$

Тогда

$$b = (X^\top X)^{-1} X^\top y.$$

Система нормальных уравнений

$$X^\top y = X^\top X b$$

для линейной модели (5.1) — это система линейных алгебраических уравнений, для решения которой можно использовать функцию `solve`:

```
solve(t(X)%*%X,t(X)%*%y) -> b; b
```

```
        [,1]
   10.485161
x1  2.976840
x2  7.349443
```

### Использование функции `R lm`

В `R` имеется специальная функция `lm` для работы с линейными моделями.

```
lm(formula, data, subset, weights, na.action,
   method = "qr", model = TRUE, x = FALSE, y = FALSE, qr = TRUE,
   singular.ok = TRUE, contrasts = NULL, offset, ...)
```

- `formula` — описание модели.
- `data` — фрейм данных, список или окружение, содержащие значения переменных, используемых в модели.
- `subset` — (необязательный) вектор, содержащий подмножество наблюдений, используемых для настройки модели.
- `weights` — (необязательный) вектор весов, используемый во взвешенном методе наименьших квадратов.
- `na.action` — функция, показывающая, что должно происходить, если данные содержат пропуски (`NA`).

Выражение для `formula` в виде `y ~ model` указывает, что отклик `y` моделируется как линейный прогноз, определённый символьной моделью `model`, состоящей из последовательности термов, разделённых операторами `+`. Сами термы содержат переменные и имена факторов, разделённых операторами `:`. Такие термы интерпретируются как взаимодействия всех переменных и факторов, встречающихся в терме.

В выражении для модели в дополнение к `+` и `:` также могут встретится другие операторы.

- `*` — оператор скрещивания (?) факторов:

  `x1*x2` интерпретируется как `x1+x2+x1:x2`.

- `^` — оператор скрещивания указанного порядка:

  `(x1+x2)^2` аналогично `(x1+x2)*(x1+x2)`.

- `%in%` — оператор, чей левый операнд вложен в правый:

  `x1+x2%in%x1` раскрывается в `x1+x1:x2`.

- `/` — оператор-сокращение:

  `x1/x2` означает `x1+x2%in%x1`.

- `-` — оператор, удалающий указанные термы:

  `(x1+x2+x3)^2 - x1:x2` аналогично `x1+x2+x3+x1:x3+x2:x3`.

Функция `I` может использоваться для ограничения части формулы, где знаки операций используются в их арифметическом смысле, например: `y ~ x1 + I(x1^2) + I(x1+x2)`.

Некоторые варианты представления для `formula`:

- `y~x` — модель парной линейной регрессии

  $$y = \beta_0 + \beta_1 x + \varepsilon.$$

  Если `y~0+x`, то

  $$y = \beta x + \varepsilon.$$

- `y~poly(x,2)` аналогично `y~X+I(x^2)`:

  $$y = \beta_0 + \beta_1 x \beta_2 x^2 + \varepsilon.$$

- `y~X` модель (множественной) линейной регрессии с матрицей независимых переменных $X$ (часто первый столбец состоит из единиц):

  $$y = X\beta + \varepsilon.$$

- `y~A` — однофакторный дисперсионный анализ модели $y$ с классами, определёнными $A$.

- `y~A+x` — однофакторный дисперсионный анализ модели $y$ с классами, определёнными $A$, и ковариатой[1] `x`.

Более подробное описание см. [28: Ch. 11: Statistical models in R].

```
lm(y~x1+x2)
```

[1] Ковариата — переменная, которая влияет на переменную отклика, но не представляет интереса для исследования.

```
Call:
lm(formula = y ~ x1 + x2)

Coefficients:
(Intercept)           x1           x2
     10.485        2.977        7.349
```

Применение к результату функции summary позволяет вывести на экран гораздо больше информации:

```
summary(lm(y~x1+x2))


Call:
lm(formula = y ~ x1 + x2)

Residuals:
    Min      1Q  Median      3Q     Max
-6.8432 -1.5716  0.1338  1.6581  6.6184

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  10.4852     0.6512   16.10   <2e-16 ***
x1            2.9768     0.1066   27.94   <2e-16 ***
x2            7.3494     0.4055   18.12   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 2.67 on 88 degrees of freedom
Multiple R-squared:  0.9252,    Adjusted R-squared:  0.9234
F-statistic: 543.8 on 2 and 88 DF,  p-value: < 2.2e-16
```

### 5.1.2 Метод максимального правдоподобия

Если известно, как распределена случайная величина $\varepsilon$ и, следовательно, известен вид выражения её функции распределения плотности вероятности $d_\varepsilon$, то можно оценить значения параметров модели (5.1), решая задачу поиска максимума функции правдоподобия:

$$L(\beta) = \prod_{i=1}^{n} d_\varepsilon(\varepsilon_i) \to \max$$

или, что более выгодно с точки зрения числовых расчётов, логарифмической функции правдоподобия:

$$\log L(\beta) = \sum_{i=1}^{n} \log\bigl(d_\varepsilon(\varepsilon_i)\bigr) \to \max\,.$$

Предположим, что $\varepsilon_i = y_i - \hat{y}_i \sim \mathcal{N}(0, \sigma_\varepsilon^2)$ и, соответственно,

$$d_\varepsilon(y_i - \hat{y}_i) = \frac{1}{\sqrt{2\pi\sigma_\varepsilon^2}} \exp\left(-\frac{(y_i - \hat{y}_i)^2}{2\sigma_\varepsilon^2}\right).$$

```
d <- function(e,s2) 1/(2*pi*s2)^0.5*exp(-e^2/(2*s2))

logL <- function(coeff)
{
  b <- coeff[-length(coeff)]
  s2 <- coeff[length(coeff)]
  e <- y-X%*%b
  sum(log(d(e,s2)))
}

optim(1:(dim(X)[2]+1),function(coeff) -logL(coeff))
```

```
$par
[1] 10.485749  2.976562  7.349135  6.896762
...
```

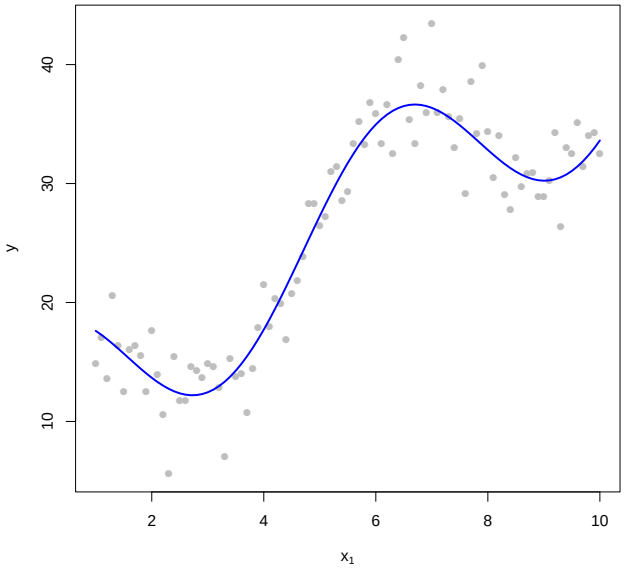
*Замечание.* Метод максимального правдоподобия можно применять для оценки параметров регрессионной модели, если функция плотности распределения ошибок $d_\varepsilon$ модели имеет максимум. Например, при $\varepsilon_i \sim \mathcal{N}(0, \sigma_\varepsilon^2)$ можно использовать оценки максимального правдоподобия, а при $\varepsilon_i \sim \mathcal{U}(-1, 1)$ — нет.

### 5.1.3 Оценка качества модели

$$y = X\beta + \varepsilon, \quad \tilde{y} = Xb, \quad e = y - \tilde{y}.$$

Для оценки качества модели будем использовать все имеющиеся данные:

```
ym <- X%*%b   # ỹ
e  <- y-ym    # e = y − ỹ

plot(x1,y,pch=16,col='gray',
     xlab=expression(x[1]),ylab='y')
lines(x1,ym,lwd=2,col='blue')
```



**Средняя абсолютная ошибка** показывает, на сколько в среднем мы будем ошибаться, используя модель:

$$\frac{1}{n}\sum_{i=1}^{n}|e_i|$$

```
mean(abs(e))                    # 2.070133
```

Для средней абсолютной ошибки нельзя сказать, большая она или маленькая. Определения «большой» или «маленький» («хороший», «плохой», «высокий», «низкий») являются относительными и их можно использовать только тогда, когда есть некий эталон, с которым можно провести сравнение: большой (или маленький) по сравнению с...

**Средняя относительная ошибка** показывает, какую долю в среднем составляет ошибка по отношению к моделируемым значениям:

$$\frac{1}{n}\sum_{i=1}^{n}\left|\frac{e_i}{y_i}\right|$$

```
mean(abs(e)/y)                  # 0.1068701
```

Средняя относительная ошибка — относительная величина и поэтому может быть оценена в терминах «большой»/«маленький». Ниже приведён вариант шкалы для оценок модели:

- ☺ $0\% \div 10\%$ — достаточно точная модель.
- 😐 $10\% \div 50\%$ — удовлетворительная точность.
- ☹ $50\% \div 100\%$ — очень грубая, неточная модель.
- Больше $100\%$ — абсолютно некачественная модель. Требует замены.

**Коэффициент детерминации** $R^2$ показывает, какая доля вариации величины $y$ описывается моделью.

$$\mathrm{TSS} = \sum_{i=1}^{n}(y_i - \bar{y})^2, \quad \mathrm{RSS} = \sum_{i=1}^{n}(\tilde{y}_i - \bar{y})^2, \quad \mathrm{ESS} = \sum_{i=1}^{n} e_i^2,$$

Оказывается, что TSS — общую (total) сумму квадратов центрированных величин $y_i - \bar{y}$ — можно разбить на две части: ту, что относится к модели — RSS (regression), и сумму квадратов ошибок — ESS (errors).

```
TSS <- sum((y-mean(y))^2)       # 8378.456
RSS <- sum((ym-mean(y))^2)      # 7232.631
ESS <- sum(e^2)                 # 638.0023
```

$$\mathrm{TSS} = \mathrm{RSS} + \mathrm{ESS} \quad \Rightarrow \quad 1 = \frac{\mathrm{RSS}}{\mathrm{TSS}} + \frac{\mathrm{ESS}}{\mathrm{TSS}}$$

Коэффициент детерминации (коэфициент определённости):

$$R^2 = \frac{\mathrm{RSS}}{\mathrm{TSS}}$$

```
R2 <- RSS/TSS; R2               # 0.8632415
```

Коэффициент неопределённости (доля вариации $y$, не описываемая моделью):

```
1-R2                            # 0.1367585
```

или

```
ESS/TSS
```

**Исправленный коэффициент детерминации** позволяет сравнивать модели с разным числом факторов $x$.

$$R^2_{\text{adj}} = 1 - (1 - R^2)\frac{n-1}{n-m-1} \quad \leqslant R^2.$$

```
1-(1-R2)*(n-1)/(n-m-1)          # 0.8601333
```

### 5.1.4 Условия Гаусса-Маркова

Если выполнены условия Гаусса-Маркова:

1. модель правильно специфицирована: $y = X\beta + \varepsilon$;
2. $\operatorname{rank} X = m + 1$ и $X$ — детерминирована (или же нет корреляции между $X$ и $\varepsilon$);
3. $\mathsf{E}\,\varepsilon = 0$; $\mathsf{Var}\,\varepsilon = \mathsf{E}\,\varepsilon_i^2 = \sigma_\varepsilon^2$; $\operatorname{Cov}(\varepsilon_i, \varepsilon_j) = 0$ при $i \neq j$,

то оценки $b$, полученные методом наименьших квадратов, являются наилучшими в классе линейных несмещённых оценок (Best Linear Unbiased Estimations).

*Замечание.* Иногда третье условие Гаусса-Маркова заменяется на более сильное: $\varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2 I_n)$.

#### Проверка значимости модели: критерий Фишера

Наблюдаемое значение в критерии Фишера показывает отношение нормированных сумм квадратов центрированных модельных значений (вариация, описываемая моделью) и ошибок (вариация, не описываемая моделью)

$$F_{\text{набл}} = \frac{\frac{1}{m}\sum_{i=1}^n (\tilde{y}_i - \bar{y})^2}{\frac{1}{n-m-1}\sum_{i=1}^n e_i^2} = \frac{\frac{1}{m}\text{RSS}}{\frac{1}{n-m-1}\text{ESS}} = \frac{n-m-1}{m}\frac{R^2}{1-R^2} \quad \sim \mathcal{F}(m, n-m-1).$$

В критерии Фишера проверяется гипотеза

$$\text{H}_0\colon \beta_1 = \cdots = \beta_m = 0.$$

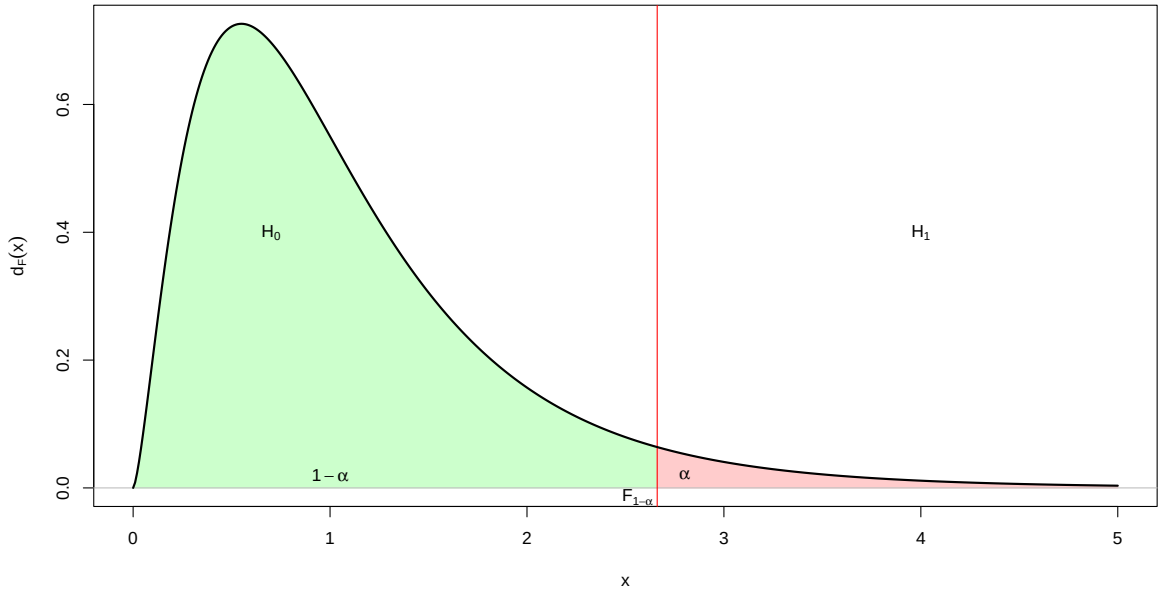
Альтернативная гипотеза:

$$\text{H}_1 = \neg\text{H}_0\colon \beta_1 \neq 0 \text{ или } \ldots \text{ или } \beta_m \neq 0.$$

Заметим, что в $\text{H}_1$ используется *неисключающее* «или». Например, если $m = 2$, то

| | $\beta_2 = 0$ | $\beta_2 \neq 0$ |
|---|---|---|
| $\beta_1 = 0$ | $\text{H}_0$ | $\text{H}_1$ |
| $\beta_1 \neq 0$ | $\text{H}_1$ | $\text{H}_1$ |

Выражение «модель незначима» означает, что значение $y$ никак не зависит от $x$. Следовательно, значения всех коэффициентов при $x$ должны быть близки к нулю (незначимые факторы), а изменение модельного значения $\tilde{y}$ относительно $\bar{y}$ — невелико по отношению к значениям ошибки модели $e$. Таким образом, если значение $F_{\text{набл}}$ достаточно мало́, то мы не может отвергнуть гипотезу $\text{H}_0$. С другой стороны, большие значения $F_{\text{набл}}$ маловероятны, когда $\text{H}_0$ истинна, поэтому $\text{H}_0$ в этом случает отвергается и принимается гипотеза $\text{H}_1$ («модель значима», $y$ зависит от $x$).




```
(n-m-1)/m * R2/(1-R2)    # F_набл = 277.735

alpha <- 0.05
qf(1-alpha,m,n-m-1)      # F_{1-α}(m, n-m-1) = 3.100069
```

Так как наблюдаемое значение $F_{\text{набл}}$ намного больше критического $F_{1-\alpha}(m, n-m-1)$, то нулевая гипотеза отклоняется и принимается критическая: модель значима.

### Проверка значимости коэффициентов: критерий Стьюдента

Если в критерии Фишера проверяется, есть ли зависимость $y$ от всех (или хотя бы от какого-нибудь) $x$, то в критерии Стьюдента проверяется значимость отдельных $x_j$.

$$\text{H}_0\colon \beta_j = 0.$$

Если $\beta_j$ равно или близко к нулю, то, очевидно, изменение соответствующего регрессора $x_j$ не оказывает влияния на $y$.

Для оценки того, близко или далеко $\beta_j$ от нуля, отношение вычисленного значения $b_j$ к его стандартной ошибке $s_{b_j}$ сравнивается с критическим значением.

$$t_{j,\text{набл}} = \frac{b_j}{s_{b_j}} \quad \sim t(n-m-1).$$

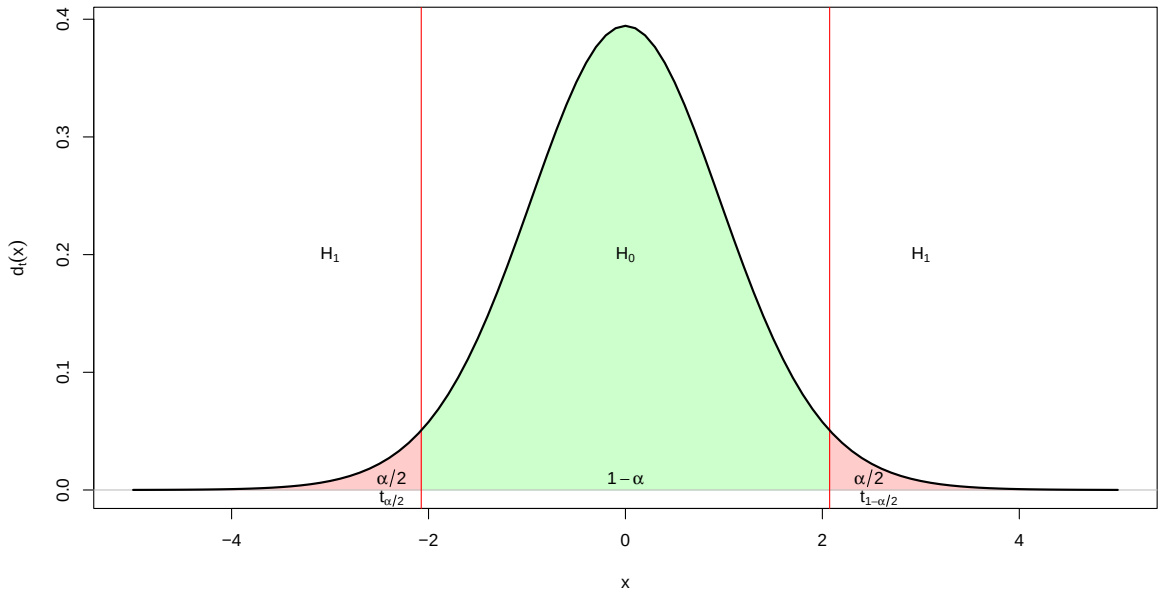
Для оценки значения стандартной ошибки $s_{b_j}$

$$s_{b_j} = s_e\sqrt{C_{j+1,j+1}}$$

требуется знать дисперсию ошибок модели

$$s_e^2 = \frac{1}{n-m-1}\sum_{i=1}^{n} e_i^2, \quad s_e = \sqrt{s_e^2}$$

и матрицу $C = (X^\top X)^{-1}$.




Так как распределение Стьюдента симметрично относительно нуля, то имеются *два* критических значения: $t_{\alpha/2}(n-m-1)$ и $t_{1-\alpha/2}(n-m-1)$. Событие $t_{j,\text{набл}} < t_{\alpha/2}(n-m-1)$ или $t_{j,\text{набл}} > t_{1-\alpha/2}(n-m-1)$ может произойти с вероятностью $\alpha$, что при $\alpha \approx 0$ требует отклонить нулевую гипотезу (слишком мала вероятность, что она истинна) и принять альтернативную: $\mathrm{H}_1\colon \beta_j \neq 0$ — коэффициент значим, так как находится достаточно далеко от нуля.

```
se <- (sum(e^2)/(n-m-1))^0.5      # s_e
C <- solve(t(X)%*%X)              # C = (X^T X)^{-1}
sb <- se*diag(C)                  # s_b
b/sb                              # b_j / s_{b_j}
```

```
        [,1]
   65.43197
x1 693.86791
x2 118.28481
```

```
qt(alpha/2,n-m-1)                 # t_{α/2}(n-m-1) = -1.98729
qt(1-alpha/2,n-m-1)               # t_{1-α/2}(n-m-1) = 1.98729
```

Так как все наблюдаемые значения находятся вне доверительного интервала

$$\big[t_{\alpha/2}(n-m-1), t_{1-\alpha/2}(n-m-1)\big],$$

то все они — значимы.

### Гетероскедастичность ошибок

Наличие у ошибок гетероскедастичности означает непостояность дисперсии — нарушение третьего условия Гаусса-Маркова. Обычно возникает из-за неоднородности исследуемых объектов: например, если рассматривается зависимость объёма выпуска продукции от производственных факторов, то *абсолютные* ошибки для мегакорпораций будут существенно превосходить ошибки, относящиеся к маленьким предприятиям: $\varepsilon_i \sim x_{j,i}$.

**Тест Голдфелда-Квандта** используется для проверки на наличие зависимости между $x_j$ и $\varepsilon$ [12: стр. 178–179].

1. Выбираем фактор $x_j$, который будем проверять на связь с $\varepsilon$, и сортируем все данные, как $x$, так и $y$, в порядке возрастания $x_j$.

   После этого выборка делится на три части: середина (примерно четверть всех наблюдений), начало ($n_1$ наблюдений) и конец ($n_2$ наблюдений).

```
o <- order(x2)                    # порядок, в котором следует выстроить x2
x1s <- x1[o]
x2s <- x2[o]
ys  <- y[o]

nn <- (n-n%/%4)%/%2
n1 <- 1:nn                        # начало
n2 <- (n-nn+1):n                  # конец
```

2. Построить две модели, используя отсортированные данные начала и конца, и вычислить для них $\mathrm{ESS}_1$ и $\mathrm{ESS}_2$.

```
lm(ys[n1]~x1s[n1]+x2s[n1]) -> f1
lm(ys[n2]~x1s[n2]+x2s[n2]) -> f2

ESS1 <- sum(f1$res^2)
ESS2 <- sum(f2$res^2)
```

3. Сравнить отношения нормированных сумм квадратов — бóльшей (например, $\mathrm{ESS}_2$) к меньшей) — с критическим значением распределения Фишера для заданного уровня значимости:

$$\frac{\mathrm{ESS}_2/(n_2-m-1)}{\mathrm{ESS}_1/(n_1-m-1)} \sim \mathcal{F}(n_2-m-1, n_1-m-1).$$

   Здесь $m$ — количество регрессоров в модели. Также предполагается наличие свободного коэффициента $\beta_0$.

```
max(ESS1,ESS2)/min(ESS1,ESS2)               # 1.135986
alpha <- 0.05
qf(1-alpha,length(n2)-m-1,length(n1)-m-1)   # 1.822132
```

Так как наблюдаемое значение меньше критического, то с увеличением $x_2$ абсолютные значения ошибки возрастают незначительно. Следовательно, гетероскедастичность отсутствует (в данном случае при выборе сортировки по возрастанию $x_2$).

Аналогичный результат можно получить, воспользовавшись функцией `gqtest` из библиотеки `lmtest`:

```
gqtest(formula, point = 0.5, fraction = 0,
  alternative = c("greater", "two.sided", "less"),
  order.by = NULL, data = list())

library(lmtest)

gqtest(y~x1+x2)
```

```
Goldfeld-Quandt test

data:  y ~ x1 + x2
GQ = 1.1311, df1 = 43, df2 = 42, p-value = 0.3454
alternative hypothesis: variance increases from segment 1 to 2
```

Удаляем четверть данных из середины после сортировки по $x_2$:

```
gqtest(y~x1+x2,order.by=x2,fraction=.25)
```

```
Goldfeld-Quandt test

data:  y ~ x1 + x2
GQ = 1.1334, df1 = 32, df2 = 31, p-value = 0.3645
alternative hypothesis: variance increases from segment 1 to 2
```

Вероятность ошибки $\approx 0.36$ — гетероскедастичности нет.

**Тест Бреуша-Пагана** используется в предположении, что величина ошибки зависит от комбинации регрессоров.

1. После оценки коэффициентов $b$ модели множественной регрессии $y = X\beta + \varepsilon$ вычисляются ошибки $e = y - Xb$.

   ```
   lm(y~x1+x2) -> f
   e <- f$res
   ```

2. Вычисляется
   $$s^2 = \frac{1}{n}\sum_{i=1}^{n} e_i^2$$
   и оцениваются коэффициенты модели
   $$\frac{e_i^2}{s^2} = \gamma_0 + \sum_{j=1}^{n'} \gamma_j x'_j + \nu_i, \tag{5.2}$$
   где $x'_j$ — независимые переменные, от которых, возможно, зависит величина ошибок $\varepsilon$.

   ```
   s2 <- sum(e^2)/n

   es <- e^2/s2
   lm(es~x1+x2) -> fe
   ```

3. Для (5.2) вычисляется сумма квадратов центрированных модельных значений RSS:
   $$\frac{1}{2}\mathrm{RSS} \quad \sim \chi^2(n').$$

   ```
   sum((fe$fit-mean(es))^2)/2      # RSS/2 =0.2358446
   alpha <- 0.05
   qchisq(1-alpha,2)               # 5.991465
   ```

Так как $\mathrm{RSS}/2 < \chi^2_{1-\alpha}(n')$, то нет связи между комбинацией $x_1$ и $x_2$ и величиной $\varepsilon$. Гетероскедастичность отсутствует.

Аналогичный результат можно получить, воспользовавшись функцией `bptest` из библиотеки `lmtest`:

```
bptest(formula, varformula = NULL, studentize = TRUE,
       data = list(), weights = NULL)
```

```
library(lmtest)
```

```
bptest(lm(y~x1+x2))
```

```
studentized Breusch-Pagan test

data:  lm(y ~ x1 + x2)
BP = 0.22448, df = 2, p-value = 0.8938
```

```
bptest(lm(y~x1+x2),studentize=FALSE)
```

```
Breusch-Pagan test

data:  lm(y ~ x1 + x2)
BP = 0.23584, df = 2, p-value = 0.8888
```

**Метод взвешенных наименьших квадратов** применяется в случае неоднородных исходных данных, например, когда требуется учесть непостоянство дисперсии.

```
wt <- 1 / lm(abs(model$residuals) ~ model$fitted.values)$fitted.values^2
```

```
wls_model <- lm(score ~ hours, data = df, weights=wt)
```

### Автокорреляция остатков

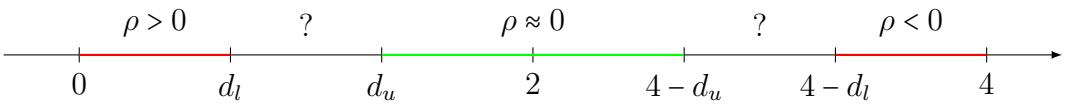
**Тест Дарбина-Уотсона** в простейшем случае используется для проверки на наличие у погрешностей $\varepsilon_i$ автокорреляци первого порядка:

$$\varepsilon_i = \rho\varepsilon_{i-1} + \nu_i, \qquad \nu_i \sim \text{i.i.d.}(0, \sigma^2_\nu).$$

1. Вычислить статистику Дарбина-Уотсона

$$\mathrm{DW} = \frac{\sum_{i=2}^n (e_i - e_{i-1})^2}{\sum_{i=1}^n e_i^2} \approx 2(1-\rho).$$

2. Сравнить DW с табличными значениями $d_l$ и $d_u$:



Если DW попадает в область неопределённости $(d_l, d_u)$ или $(4-d_u, 4-d_l)$ то, конечно, автокорреляции может не быть, но безопаснее и надёжнее считать, что она есть.

```
lm(y~x1+x2) -> f
e <- f$res

sum((e[-1]-e[-length(e)])^2)/sum(e^2)
```

```
2.145326
```

В библиотеке `car` имеется функция `durbinWatsonTest`, которая выполняет те же действия автоматически.

```
library(car)

durbinWatsonTest(lm(y~x1+x2))
```

```
 lag Autocorrelation D-W Statistic p-value
   1     -0.07977716      2.145326   0.622
 Alternative hypothesis: rho != 0
```

Значение `p-value = 0.622` ($\gg 0.05$) говорит о малой вероятности наличия автокорреляции остатков.

**Коррекция данных** Пусть

$$y_i = \beta_0 + \sum_{j=1}^{m} \beta_j x_{j,i} + \varepsilon_i. \tag{5.3}$$

Если предполагается наличие у ошибок автокорреляции первого порядка:

$$\varepsilon_i = \rho\varepsilon_{i-1} + \nu_i, \qquad \nu_i \sim \text{i.i.d.}(0, \sigma_\nu^2),$$

то (5.3) заменяется на

$$\hat{y}_i = \hat{\beta}_0 + \sum_{j=1}^{m} \beta_j \hat{x}_{j,i} + \nu_i, \quad i = 1, \ldots, n,$$

где

$$\hat{y}_i = \begin{cases} \sqrt{1-r^2}y_1, & i = 1, \\ y_i - ry_{i-1}, & i = 2, \ldots, n, \end{cases} \qquad \hat{x}_{j,i} = \begin{cases} \sqrt{1-r^2}x_{j,1}, & i = 1, \\ x_{j,i} - ry_{j,i-1}, & i = 2, \ldots, n. \end{cases}$$

Так как $\text{DW} \approx 2(1-\rho)$, то $\rho \approx 1 - \text{DW}/2$[2].

```
n <- length(x1)
sum((e[-1]-e[-n])^2)/sum(e^2) -> DW
r <- 1-DW/2

X1 <- c((1-r^2)*x1[1],x1[-1]-r*x1[-n])
X2 <- c((1-r^2)*x2[1],x2[-1]-r*x2[-n])
Y  <- c((1-r^2)*y[1],  y[-1]-r*y[-n])
lm(Y~X1+X2)
```

[2]Значение $1-\text{DW}/2$ может быть использовано как начальное приближение при оценке значения $\rho$ методом Кохрейна-Орката (см. [12: стр. 187]). Также более точная по сравнению с $1-\text{DW}/2$ оценка значения $\rho$ может быть получена с помощью сеточного поиска в методе Хилдрета-Лу (см. [12: стр. 188])

```
Call:
lm(formula = Y ~ X1 + X2)

Coefficients:
(Intercept)           X1           X2
     11.200        2.983        7.339
```

```
durbinWatsonTest(lm(Y~X1+X2))
```

```
lag Autocorrelation D-W Statistic p-value
   1     -0.01226242      2.003446   0.884
 Alternative hypothesis: rho != 0
```

Вероятность наличия автокорреляции остатков стала ещё меньше.

### 5.1.5 Прогнозирование

#### Вычисление прогноза вручную

Пусть для

$$y = \beta_0 + \sum_{j=1}^{m} \beta_j x_j + \varepsilon \quad \text{или} \quad y = X\beta + \varepsilon$$

выполнена оценка значений коэффициентов и

$$\tilde{y} = Xb, \quad e = y - \tilde{y}.$$

Точечный прогноз $\tilde{y}^*$ при $x^*$ (вектор-строка) вычисляется не просто, а очень просто:

$$\tilde{y}^* = x^* b.$$

Для оценки качества прогноза оценим

1. среднеквадратическое отклонение для ошибки модели

$$s_e = \sqrt{\frac{1}{n-m-1}\sum_{i=1}^{n} e_i};$$

2. стандартную ошибку прогноза:

$$s_{\tilde{y}^*} = s_e \sqrt{x^*(X^\top X)^{-1}(x^*)^\top};$$

3. границы доверительного интервала при уровне значимости $0 < \alpha < 1$:

$$\tilde{y}^* + s_{\tilde{y}^*} t_{\alpha/2}(n-m-1) \leqslant y^* \leqslant \tilde{y}^* + s_{\tilde{y}^*} t_{1-\alpha/2}(n-m-1).$$

   Здесь $t_{\alpha/2}(n-m-1)$ — $\alpha/2$-квантиль для $t$-распределения Стьюдента при $n-m-1$ степенях свободы.

```
yp <- xp%*%b

e <- y-ym
se <- (sum(e^2)/(n-m-1))^0.5
sy <- se*(xp%*%solve(t(X)%*%X)%*%xp)^0.5
alpha <- 0.05

c(yp+sy*qt(alpha/2,n-m-1),yp,yp+sy*qt(1-alpha/2,n-m-1))
```

**Использование функции `predict`**

```
lm(y ~ x) -> f

# точечный прогноз
predict(f,list(x=1500))
# точечный прогноз и доверительный интервал для α = 0.95
predict(f,list(x=1500),interval='confidence')
# точечный прогноз и доверительный интервал для α = 0.99
predict(f,list(x=1500),interval='confidence',level=0.99)
```

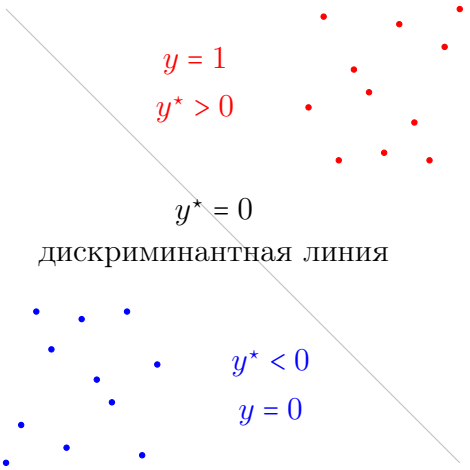
## 5.2 Модель бинарного выбора

Пусть имеется набор данных $(x_i, y_i)$, $i = 1, \ldots, n$, где каждому $x$ приписан маркер $y$, имеющий *качественное бинарное* значение. Вне зависимости от исходных значений $y$ (мужчина и женщина, man и woman, Mann и Frau, homme и femme или 男 и 女) будем использовать кодировку 0 для первого значения $y$ и 1 для второго.

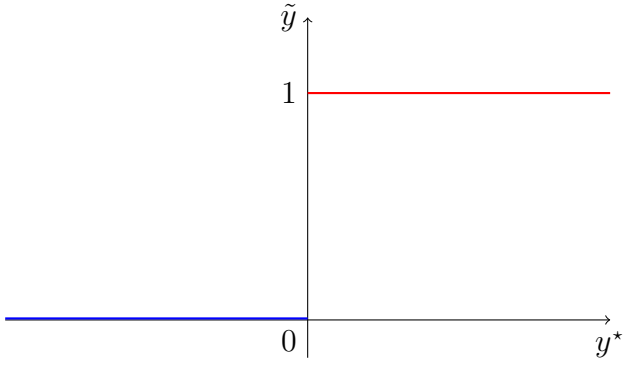
Предполагая, что значение маркера $y$ определяется значением $x$: $y \sim x_1, \ldots, x_m$, попытаемся построить такую *дискриминантную функцию* $y^\star$, что

$$y^\star = \beta_0 + \sum_{j=1}^{m} \beta_j x_j, \qquad \tilde{y} = \begin{cases} 0, & y^\star < 0 \\ 1, & y^\star > 0 \end{cases}$$

Здесь $\tilde{y}$ — результат, выдаваемый построенной моделью бинарного выбора.

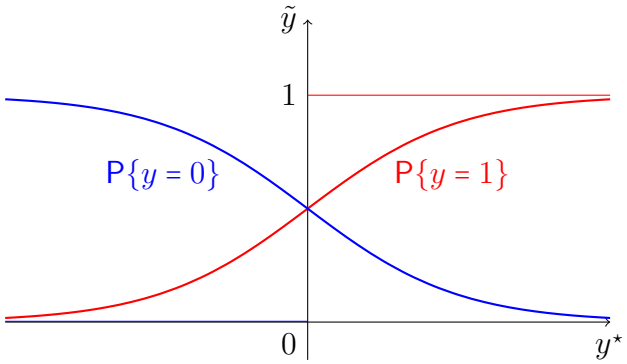



Заметим, что в построенной модели результат зависит только от знака $y^\star$: если $y^\star > 0$, то $\tilde{y} = 1$ независимо от того, $y^\star = 0.0001$ или $y^\star = 10000$.



Нелинейное преобразование $\Phi\colon \mathbb{R} \to (0,1)$ даёт возможность рассматривать преобразованные значения $y^\star$ как вероятности события $y = 1$:

$$\Phi(y^\star) = \mathsf{P}\{y = 1\}.$$




Заметим, что абсолютную величину разницы между $\mathsf{P}\{y = 1\}$ и $\mathsf{P}\{y = 0\}$ можно использовать в качестве оценки уверенности к решению, выдаваемому классификатором:

$$\mathrm{conf}\{y = \tilde{y}\} = \big|\mathsf{P}\{y = 1\} - \mathsf{P}\{y = 0\}\big|.$$

Вместо $y^\star$ можно использовать нормализованные значения $s_{y^\star}$, но при этом возможна более широкая размытая граница между классифицируемыми множествами.

Обычно для преобразования $\Phi$ используется

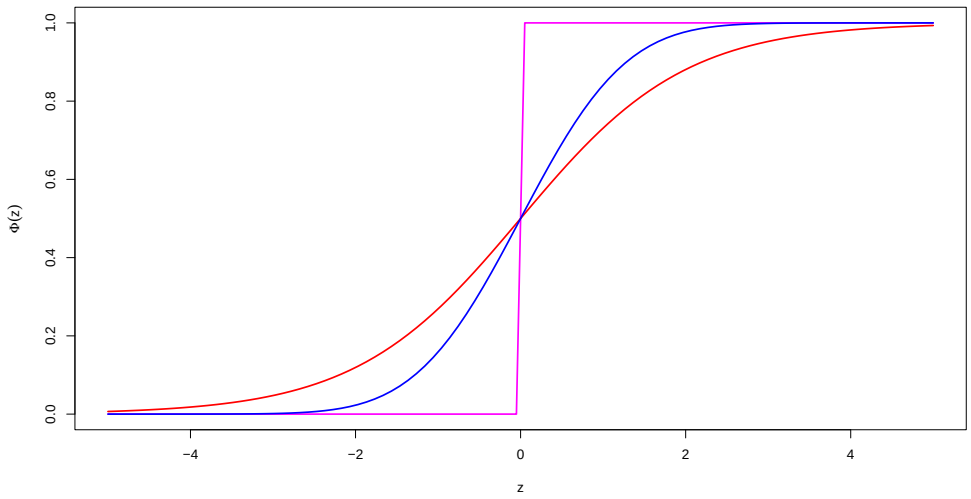
- логистическая функция (логит-модель):

$$\Phi(z) = \frac{1}{1 + \exp(-z)}$$

- функция распределения вероятностей стандартного нормального распределения (пробит-модель):

$$\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} \exp\left(-\frac{s^2}{2}\right) ds.$$

```
z <- seq(from=-5,to=5,length=100)
y <- rep(0,length(z)); y[z>=0] <- 1

phi1 <- 1/(1+exp(-z))     # логит
phi2 <- pnorm(z)          # пробит

plot(z,y,type='l',col='magenta',lwd=2,ylab=expression(Phi(z)))
lines(z,phi1,col='red',lwd=2)
lines(z,phi2,col='blue',lwd=2)
```




Для оценки коэффициентов дискриминантной функции обычно используется метод максимального правдоподобия: выбираются значения коэффициентов, при которых функция правдоподобия:

$$L = \prod_{i=1}^{n} \mathsf{P}\{\tilde{y} = y_i\} = \prod_{y_i=0} \mathsf{P}\{\tilde{y}_i = 0\} \times \prod_{y_i=1} \mathsf{P}\{\tilde{y}_i = 1\} = \prod_{y_i=0} \bigl(1 - \Phi(y_i^\star / s_{y^\star})\bigr) \times \prod_{y_i=1} \Phi(y_i^\star / s_{y^\star})\bigr)$$

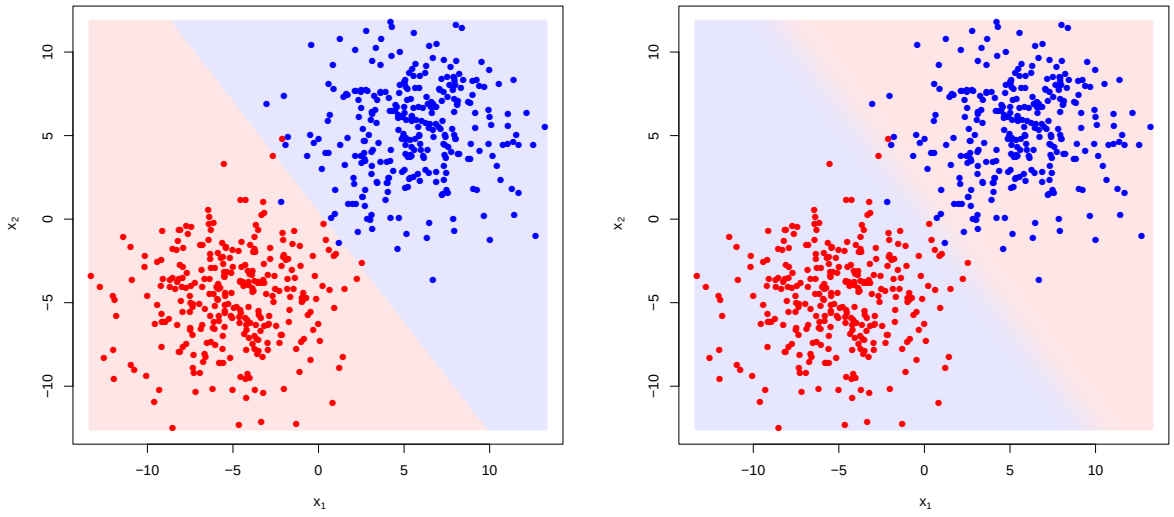
или логарифмическая функция правдоподобия:

$$\log L = \sum_{y_i=0} \log\bigl(1 - \Phi(y_i^\star / s_{y^\star})\bigr) + \sum_{y_i=1} \log \Phi(y_i^\star / s_{y^\star})\bigr)$$

принимают максимальные значения.

```
# ===================================
# обучающее множество
n <- 300

s <- 3
x1 <- c(rnorm(n,mean=-5,sd=s),rnorm(n,mean=5,sd=s))
x2 <- c(rnorm(n,mean=-5,sd=s),rnorm(n,mean=5,sd=s))
y <- c(rep(0,n),rep(1,n))

# для линейной дискриминантной функции
X <- cbind(1,x1,x2)

# ===================================
# метод максимального правдоподобия
logL <- function(b)
{
  yz <- X%*%b
  phi <- 1/(1+exp(-yz))
  sum(log(1-phi[y==0])) + sum(log(phi[y==1]))
}
# начинаем поиск со случайной точки
optim(runif(dim(X)[2],min=-1,max=1),
      function(b) -logL(b))$par -> B
```

```
# ===================================
# визуализация результата

X1 <- seq(from=min(x1),to=max(x1),length=200)
X2 <- seq(from=min(x2),to=max(x2),length=200)
# перечисление всех точек сетки X1 × X2
XX <- c()
for (x in X1) XX <- rbind(XX,cbind(1,x,X2))

# ===================================
# значения дискриминантной функции: чёткая граница
yz <- XX%*%B
# холст
plot(x1,x2,type='n',xlab=expression(~x[1]),ylab=expression(~x[2]))
# фон
points(XX[yz<0,2],XX[yz<0,3],pch=20,col=rgb(1,.9,.9))
points(XX[yz>=0,2],XX[yz>=0,3],pch=20,col=rgb(.9,.9,1))
# обучающее множество
points(x1[y==0],x2[y==0],pch=16,col='red')
points(x1[y==1],x2[y==1],pch=16,col='blue')

# ===================================
# оценка значений вероятностей P{y = 1}: размытая граница
phi <- 1/(1+exp(-yz))
# холст
plot(x1,x2,type='n',xlab=expression(~x[1]),ylab=expression(~x[2]))
# фон
points(XX[,2],XX[,3],pch=20,col=rgb(.9+phi*0.1,.9,.9+(1-phi)*0.1))
# обучающее множество
points(x1[y==0],x2[y==0],pch=16,col='red')
points(x1[y==1],x2[y==1],pch=16,col='blue')
```




Реже применяемый, но, возможно, более вычислительно устойчивый — метод наименьших квадратов:

$$\sum_{i=1}^{n}\big(y_i - \Phi(y_i^\star / s_{y^\star})\big)^2 \to \min$$

```
# метод наименьших квадратов
ESS <- function(b)
{
  ym <- X%*%b
  phi <- 1/(1+exp(-ym))
  sum((y-phi)^2)
}
optim(runif(dim(X)[2],min=-1,max=1),ESS)$par -> B
```

Для оценок коэффициентов $y^\star$ можно использовать встроенную функцию glm.

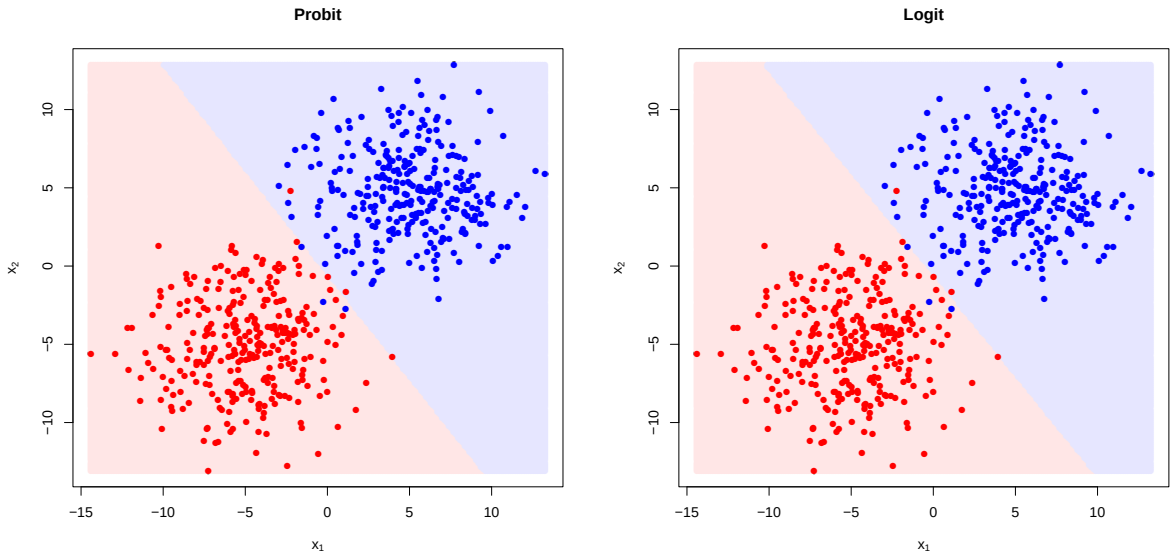
```
f1 <- glm(y~x1+x2,family=binomial(link='probit'))
f2 <- glm(y~x1+x2,family=binomial(link='logit'))

f1
```

```
Call:  glm(formula = y ~ x1 + x2, family = binomial(link = "probit"))

Coefficients:
(Intercept)           x1           x2
     0.5043       2.0054       1.4940

Degrees of Freedom: 599 Total (i.e. Null);  597 Residual
Null Deviance:      831.8
Residual Deviance: 8.039        AIC: 14.04
```

```
f2
```

```
Call:  glm(formula = y ~ x1 + x2, family = binomial(link = "logit"))

Coefficients:
(Intercept)           x1           x2
     0.9276       3.5022       2.5902

Degrees of Freedom: 599 Total (i.e. Null);  597 Residual
Null Deviance:      831.8
Residual Deviance: 8.252        AIC: 14.25
```

```
plot(x1,x2,main='Probit',xlab=expression(x[1]),ylab=expression(x[2]))
for (t1 in T1) for (t2 in T2)
{
  ym <- predict.glm(f1,newdata=list(x1=t1,x2=t2))
  if (ym < 0)
    points(t1,t2,pch=16,col=rgb(1,.9,.9))
  else
    points(t1,t2,pch=16,col=rgb(.9,.9,1))
}

points(x1[y==0],x2[y==0],pch=16,col='red')
points(x1[y==1],x2[y==1],pch=16,col='blue')
```




## 5.3 Динамические модели

В динамических моделях предполагается, что значение отклика $y_i$ зависит не только от текущего значения регрессоров $x_i$, но также от предыдущих значений $x_{i-j}$ и/или $y_{i-j}$.

Для обработки данных, представленных в виде временны́х рядов, используются функции из библиотеки `tseries`.

### 5.3.1 Модель Бокса-Дженкинса

В общем виде (и при это очень формально) модель Бокса-Дженкинса (см. [2]) может быть представлена в виде операторного уравнения

$$\theta_p(\mathsf{L})\Theta_P(\mathsf{L}^s)\Delta_d(\mathsf{L})\Delta_D^s(\mathsf{L})y_i = \phi_q(\mathsf{L})\Phi_Q(\mathsf{L}^s)\varepsilon_i, \quad \varepsilon_i \sim \text{i.i.d.}(0, \sigma_\varepsilon^2).$$

Здесь

- $\mathsf{L}$ — оператор запаздывания (лаговый оператор, оператор сдвига назад):

$$\mathsf{L}\, y_i = y_{i-1}, \qquad \mathsf{L}^k\, y_i = y_{i-k}.$$

- $\theta_p$ — авторегрессионная часть модели.
- $\phi_p$ — часть модели, отвечающая за скользящую среднюю.
- $s$ — сезонный лаг (обычно $s = 12$, если единица времени — месяц).
- $\Theta_P$ и $\Phi_Q$ — сезонные авторегрессия и скользящее среднее.
- $\Delta$ — разностный оператор:

$$\Delta y_i = (1 - \mathsf{L})y_i, \quad \Delta_d y_i = (1 - \mathsf{L})^d y_i, \quad \Delta_d^s y_i = (1 - \mathsf{L}^s)^d y_i.$$

Модель Бокса-Дженкинса в `R` реализована в виде функции `arima`

```
arima(x, order = c(0L, 0L, 0L),
      seasonal = list(order = c(0L, 0L, 0L), period = NA),
      xreg = NULL, include.mean = TRUE,
      transform.pars = TRUE,
      fixed = NULL, init = NULL,
      method = c("CSS-ML", "ML", "CSS"), n.cond,
      SSinit = c("Gardner1980", "Rossignol2011"),
      optim.method = "BFGS",
      optim.control = list(), kappa = 1e6)
```

### ARMA$(p, q)$

Простейший вариант модели, описывающий слабо стационарный временной ряд $\{y_i\}$[3].

$$y_i = \gamma + \underbrace{\sum_{j=1}^{p} \alpha_j y_{i-j}}_{\mathrm{AR}(p)} + \varepsilon_i + \underbrace{\sum_{j=1}^{q} \beta_j \varepsilon_{i-j}}_{\mathrm{MA}(q)}, \quad \varepsilon_i \sim \text{i.i.d.}(0, \sigma_\varepsilon^2).$$

Значения $p$ и $q$ определяют *порядок* модели.

Перенесём все компоненты модели, содержащие $y$ в левую часть:

$$y_i - \sum_{j=1}^{p} \alpha_j y_{i-j} = \gamma + \varepsilon_i + \sum_{j=1}^{q} \beta_j \varepsilon_{i-j}, \quad \varepsilon_i \sim \text{i.i.d.}(0, \sigma_\varepsilon^2).$$

Заменив лаговые переменные $y_{i-j}$ на $\mathsf{L}^j y_i$ и приведя подобные, получим

$$\Big(I - \sum_{j=1}^{p} \alpha_j \mathsf{L}^j\Big) y_i = \gamma + \Big(I + \sum_{j=1}^{q} \beta_j \mathsf{L}^j\Big) \varepsilon_i, \quad \varepsilon_i \sim \text{i.i.d.}(0, \sigma_\varepsilon^2)$$

или

$$\theta_p(\mathsf{L}) y_i = \gamma + \phi_q(\mathsf{L}) \varepsilon_i$$

где $\theta_p(\mathsf{L})$ и $\phi_q(\mathsf{L})$ — операторые многочлены:

$$\theta_p(\mathsf{L}) = I - \alpha_1 \mathsf{L} - \alpha_2 \mathsf{L}^2 - \cdots - \alpha_p \mathsf{L}^p,$$
$$\phi_q(\mathsf{L}) = I + \beta_1 \mathsf{L} + \beta_2 \mathsf{L}^2 + \cdots + \beta_q \mathsf{L}^q.$$

**Пример.** *Белый шум*

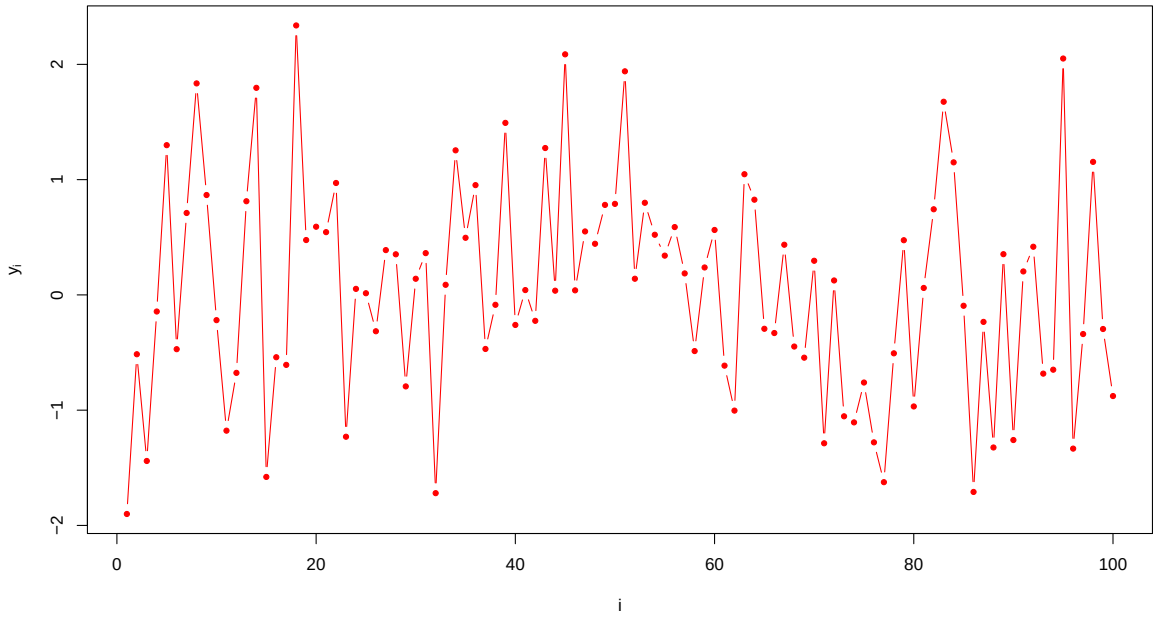
$$y_i = \varepsilon_i, \quad \varepsilon_i \sim \text{i.i.d.}(0, \sigma_\varepsilon^2).$$

Формально, белый шум не является процессом, описываемым моделью ARMA, так как память в нём полностью отсутствует: $p = q = 0$. Но $y_i = \varepsilon_i \sim \text{i.i.d.}(0, \sigma_\varepsilon^2)$ и поэтому белый шум — слабо стационарный (и даже сильно стационарный[4]) процесс.

```
y <- rnorm(n)
plot(1:n,y,type='b',pch=20,col='red',xlab='i',ylab=expression(y[i]))
```

[3] Временной ряд $\{y_i\}$ называется слабо стационарным, если на любом временно́м интервале $\mathsf{E} y_i = \mu$ и $\mathsf{Var}\, y_i = \sigma^2$ — константы, а $\mathrm{Cov}(y_i, y_{i+k})$ зависит только от $k$.

[4] Временной ряд $\{y_i\}$ называется сильно стационарным, если на любом временно́м интервале распределение $y$ одно и то же с одними и теми же значениями параметров.




❑

**Пример.** *Авторегрессионный процесс первого порядка:*

$$y_i = \gamma + \alpha_1 y_{i-1} + \varepsilon_i, \quad \varepsilon_i \sim \text{i.i.d.}(0, \sigma_\varepsilon^2).$$

Так как предполагается слабая стационарность процесса, то

$$\forall i\colon \mathsf{E}\, y_i = \mathsf{E}\, y_{i-1}$$

и, следовательно,

$$\mathsf{E}\, y_i = \frac{\gamma}{1-\alpha_1}, \quad -1 < \alpha_1 < 1.$$

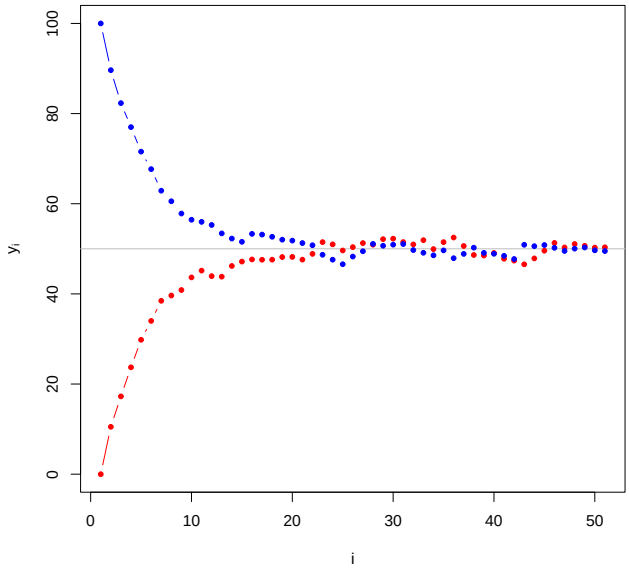
```
n <- 50
e <- array(rnorm(2*n),dim=c(2,n))

gamma <- 10
alpha <- 0.8

y <- array(dim=c(2,n+1))
y[,1] <- c(0,100)
for (i in 1:n) y[,i+1] <- gamma+alpha*y[,i]+e[,i]
```

Для генерирования значений также можно использовать функцию `arima.sim`:

```
arima.sim(model, n, rand.gen = rnorm, innov = rand.gen(n, ...),
          n.start = NA, start.innov = rand.gen(n.start, ...),
          ...)

n  <- 50
gamma <- 10
alpha <- 0.8
```

```
y0 <- 0
c(y0,arima.sim(n=n,list(ar=alpha),
               n.start=1,start.innov=y0,
               innov=rnorm(n,mean=gamma))) -> y

y0 <- 100
rbind(y,
      c(y0,arima.sim(n=n,list(ar=alpha),
           n.start=1,start.innov=y0,
           innov=rnorm(n,mean=gamma)))) -> y

# Визуализация:
plot(c(1,n+1),c(min(y),max(y)),type='n',xlab='i',ylab=expression(y[i]))
abline(h=gamma/(1-alpha),col='gray')
lines(y[1,],type='b',pch=20,col='red')
lines(y[2,],type='b',pch=20,col='blue')
```

При любых начальных значениях $y_0$ процесс после некоторого переходного периода сходится к одному и тому же значению $\gamma/(1-\alpha_1)$ (предполагается, что $-1 < \alpha_1 < 1$!).




❑

**ARIMA**$(p,d,q)$

Модель, описывающая тренд-стационарный временной ряд $\{y_i\}$, то есть разности $\{\Delta_d y_i\}$ образуют слабо стационарный временной ряд.

$$\Delta_d y_i = \sum_{j=1}^{p} \alpha_j \Delta_d y_{i-j} + \varepsilon_i + \sum_{j=1}^{q} \beta_j \varepsilon_{i-j}, \quad \varepsilon_i \sim \text{i.i.d.}(0, \sigma_\varepsilon^2).$$

**Пример.** *Случайное блуждание*

$$y_i = y_{i-1} + \varepsilon_i, \quad \varepsilon_i \sim \text{i.i.d.}(0, \sigma_\varepsilon^2).$$

Первая разность процесса случайного блуждания является белым шумом:

$$\Delta y_t = \varepsilon_t.$$

❑

**Пример.** *Уровни озера Гурон (в футах), измеренные с 1875 по 1972 гг.*

```
LakeHuron
```

```
Time Series:
Start = 1875
End = 1972
Frequency = 1
 [1] 580.38 581.86 580.97 580.80 579.79 580.39 580.42 580.82 581.40 581.32
...
[91] 576.80 577.68 578.38 578.52 579.74 579.31 579.89 579.96
```

```
plot(LakeHuron)
```


Наличие единичных корней проверяется с помощью теста Дики-Фуллера `adf.test` из библиотеки `tseries`:

```
library(tseries)
adf.test(LakeHuron)
```

```
        Augmented Dickey-Fuller Test
data:  LakeHuron
Dickey-Fuller = -2.7796, Lag order = 4, p-value = 0.254
alternative hypothesis: stationary
```

Вероятность наличия единичного корня — 0.254 (в этом случае временной ряд описывается моделью случайного блуждания).

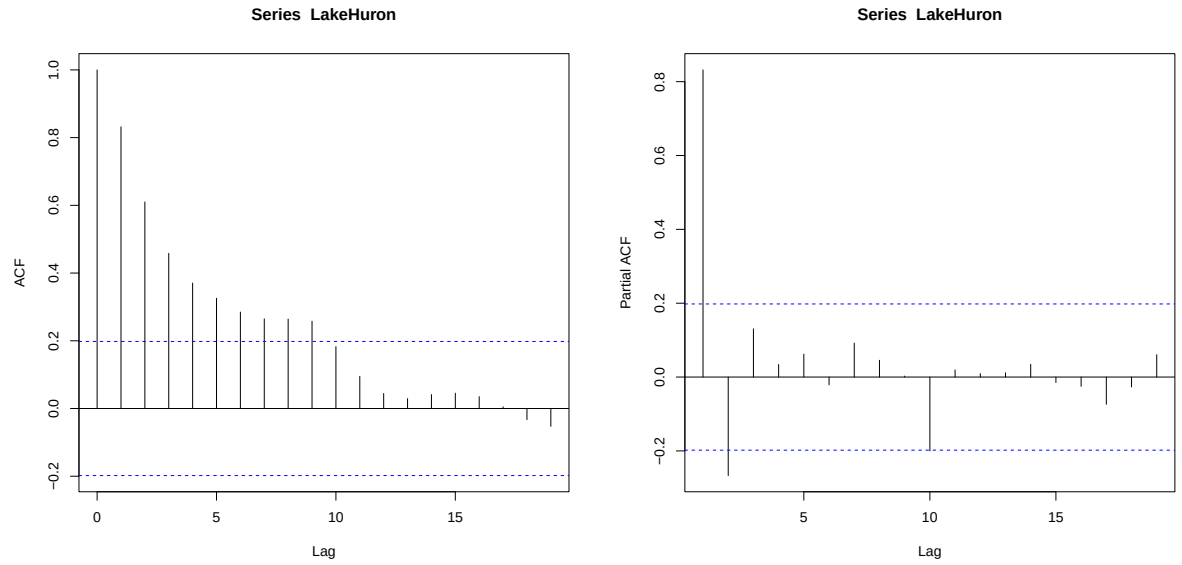
```
adf.test(diff(LakeHuron)) # Тест для ∇y_t
```

```
        Augmented Dickey-Fuller Test
data:  diff(LakeHuron)
Dickey-Fuller = -5.4687, Lag order = 4, p-value = 0.01
alternative hypothesis: stationary
Предупреждение
In adf.test(diff(LakeHuron)) : p-value smaller than printed p-value
```

Взятие первой разности удаляет тренд.

Автокорреляционная (ACF) функция вычисляет значения парных корреляций между элементами временного ряда $y_t$ и $y_{t-k}$. Частная автокорреляционная (PACF) функция показывает *чистую* силу связи между элементами ряда $y_t$ и $y_{t-k}$, из которой удалены неявные влияния других элементов $y_{t-l}$, где $l \neq 0$ и $l \neq k$.

```
acf(LakeHuron)
pacf(LakeHuron)
```




Простейшая авторегрессионная модель первого порядка AR(1):

$$y_t = \beta_0 + \beta_1 y_{t-1} + \varepsilon_t, \qquad \varepsilon_t \sim iid(0, \sigma^2).$$

```
arima(LakeHuron,order=c(1,0,0))
```

```
Call:
arima(x = LakeHuron, order = c(1, 0, 0))
Coefficients:
         ar1  intercept
      0.8375   579.1153
s.e.  0.0538     0.4240
sigma^2 estimated as 0.5093:  log likelihood = -106.6,  aic = 219.2
```

Здесь $b_1 = 0.8375$ ($s_{b_1} = 0.0538$), $b_0 = 579.1153$ ($s_{b_0} = 0.4240$) — оценки значений коэффициентов (и стандартные ошибки оценок); дисперсия остатков $s_e^2 = 0.5093$; значение логарифмической функции правдоподобия $-106.6$. `AIC`$= 219.2$ — значение информационного критерия Акаике, объединяющего сложность модели и её точность (чем меньше значение коэффициента, тем лучше)

$$\text{AIC} = -L + km.$$

Здесь $m$ — количество оцениваемых параметров; $L$ — значение логарифмической функции правдоподобия построенной модели; $k = 2$ (обычно для критерия Акаике) или $k = \log(n)$ (критерий Шварца), где $n$ — количество наблюдений.

Авторегрессионная модель второго порядка AR(2):

```
arima(LakeHuron,order=c(2,0,0))
```

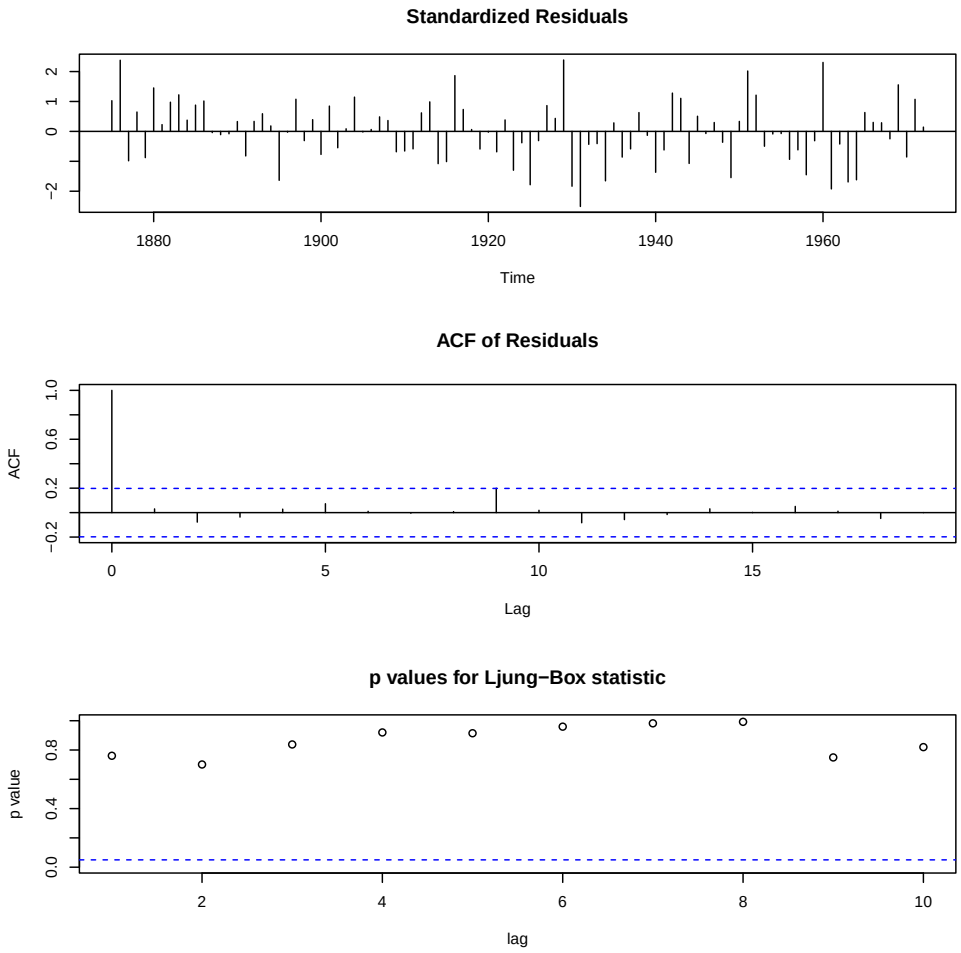
```
Call:
arima(x = LakeHuron, order = c(2, 0, 0))
Coefficients:
         ar1      ar2  intercept
      1.0436  -0.2495   579.0473
s.e.  0.0983   0.1008     0.3319
sigma^2 estimated as 0.4788:  log likelihood = -103.63,  aic = 215.27
```

$\mathrm{AIC}_{\mathrm{AR}(2)} < \mathrm{AIC}_{\mathrm{AR}(1)}$ — вторая модель лучше первой (незначительное увеличение сложности компенсируется существенным снижением погрешности).

Графический анализ результатов моделирования:

```
tsdiag(arima(LakeHuron,order=c(2,0,0)))
```

Остатки, автокорреляционная функция остатков и вероятности для статистики Льюнга-Бокса

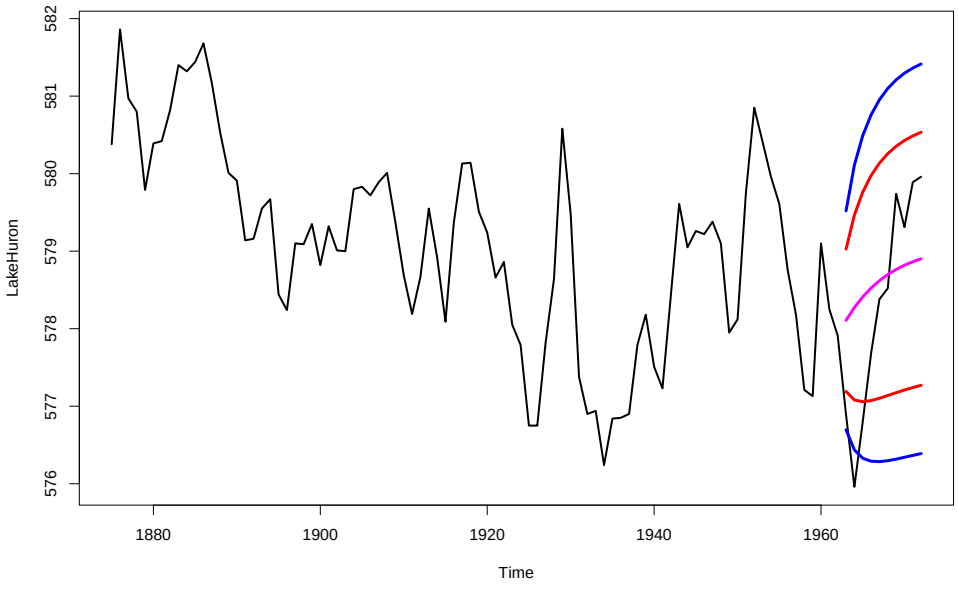



**Прогнозирование** Оценим последние $n_1$ значения, построив модель по первым $n_2 = n - n_1$ наблюдениям:

```
n <- length(LakeHuron)
n1 <- 10      # количество прогнозных значений
n2 <- n-n1    # данные для настройки модели
f <- arima(LakeHuron[1:n2],order=c(1,0,0))

predict(f,n.ahead=n1) -> p; p
```

```
$pred
Time Series:
Start = 89
End = 98
Frequency = 1
 [1] 578.1082 578.2730 578.4101 578.5242 578.6191 578.6981 578.7638 578.8184
 [9] 578.8639 578.9017
$se
Time Series:
Start = 89
End = 98
Frequency = 1
 [1] 0.7104049 0.9240977 1.0467527 1.1238299 1.1742156 1.2078570 1.2306016
 [8] 1.2460998 1.2567141 1.2640080
```

```
# Интервал прогнозирования
T <- (end(LakeHuron)[1]-n1+1):end(LakeHuron)[1]
# Критические значения t-распределения
t1 <- qt(0.975,n2-2)
t2 <- qt(0.9,n2-2)

# График результатов
plot(LakeHuron,lwd=2)                          # Исходные данные
lines(T,p$pred,lwd=3,col="magenta")            # Прогноз (пурпурная линия)
# Доверительная воронка (вероятность 95%, синий цвет)
lines(T,p$pred+t1*p$se,lwd=3,col="blue")
lines(T,p$pred-t1*p$se,lwd=3,col="blue")
# Доверительная воронка (вероятность 80%, красный цвет)
lines(T,p$pred+t2*p$se,lwd=3,col="red")
lines(T,p$pred-t2*p$se,lwd=3,col="red")
```




❑

**SARIMA**$(p,d,q)(P,D,Q)_s$

Модель SARIMA позволяет учесть эффекты сезонности.

**Пример.** *Объём авиаперевозок пассажиров по месяцам с 1949 по 1960 гг.*

```
AirPassengers
```

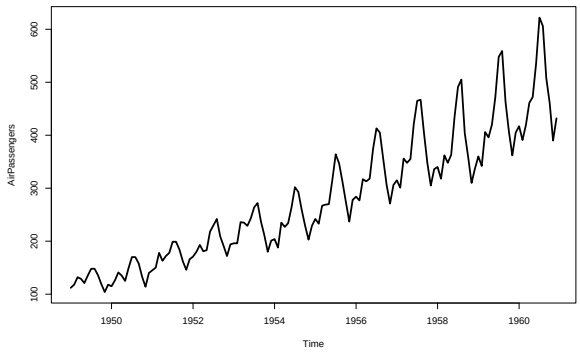
```
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1949 112 118 132 129 121 135 148 148 136 119 104 118
...
1960 417 391 419 461 472 535 622 606 508 461 390 432
```

Начало, конец и частота (количество наблюдений в единицу времени)

```
start(AirPassengers); end(AirPassengers); frequency(AirPassengers)
```

```
[1] 1949    1
[1] 1960   12
[1] 12
```

```
plot(AirPassengers,lwd=3)
```




Декомпозиция временного ряда — выделение тренда, сезонной составляющей и случайных остатков — с помощью функции `decompose`:

```
decompose(AirPassengers)
```

Графики для аддитивной модели (рис. 5.1):

$$y_t = T_t + S_t + \varepsilon_t$$

```
plot(decompose(AirPassengers,type="additive"))
```

и мультипликативной модели (рис. 5.2):

$$y_t = T_t \times S_t \times \varepsilon_t$$

Здесь $T$ — тренд, $S$ — сезонная составляющая, $\varepsilon$ — случайные остатки.

```
plot(decompose(AirPassengers,type="multiplicative"))
```

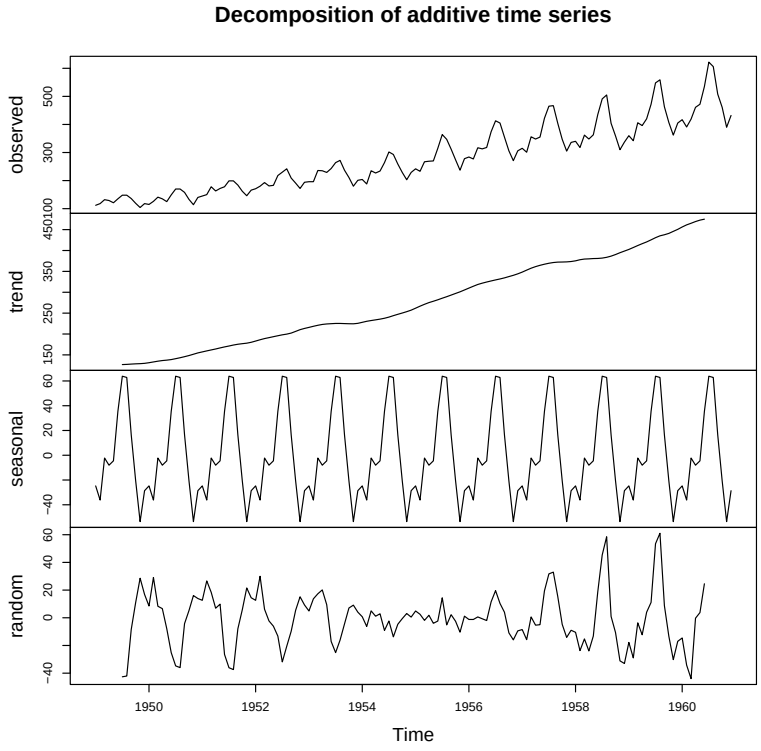



Рис. 5.1: Аддитивная сезонная составляющая

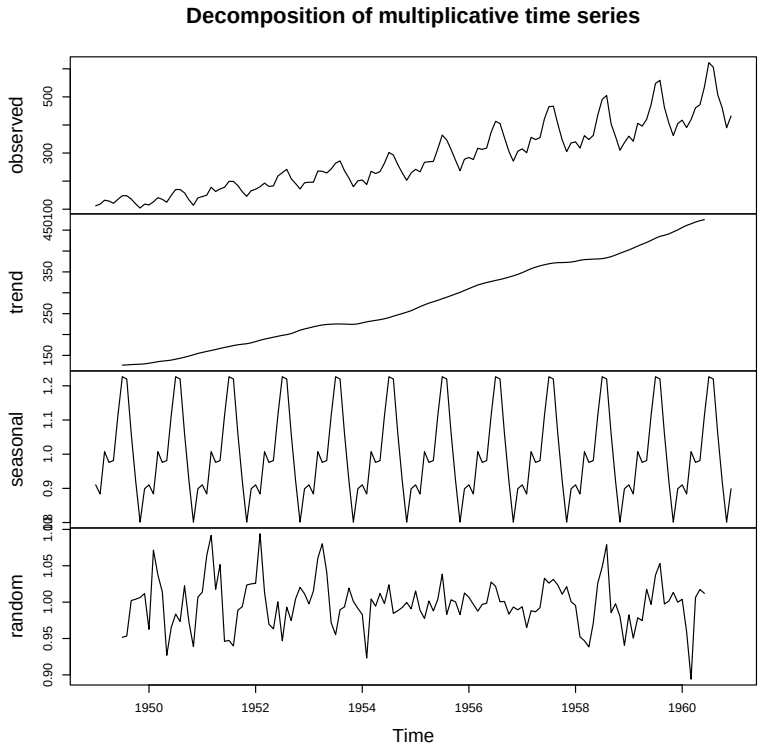



Рис. 5.2: Мультипликативная сезонная составляющая

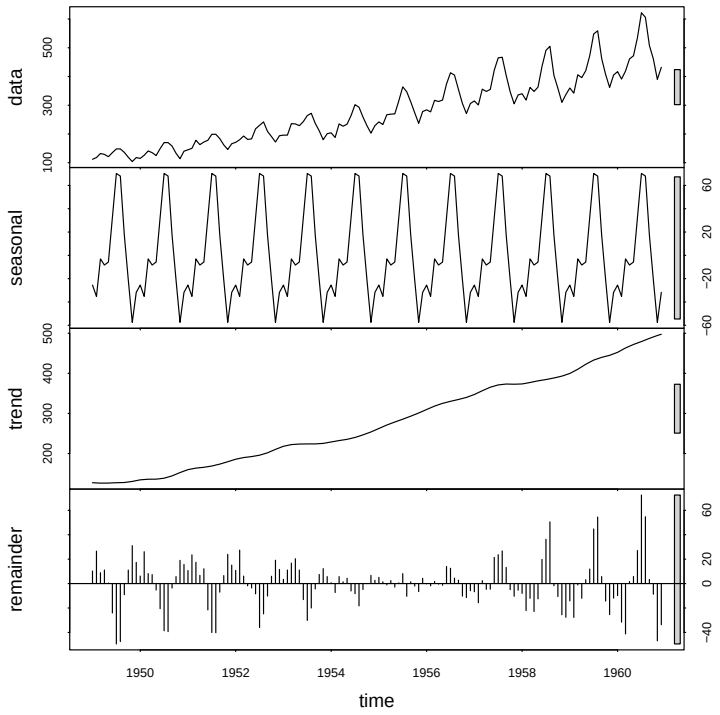
Сезонная составляющая предполагается периодической функцией $S_t = S_{t+s}$, в реальности же амплитуда сезонных изменений со временем увеличивается, поэтому случайная компонента демонстрирует неслучайное поведение.

Декомпозиция данных с помощью функции `stl`:

```
stl(AirPassengers,s.window="periodic") -> d
summary(d)
```

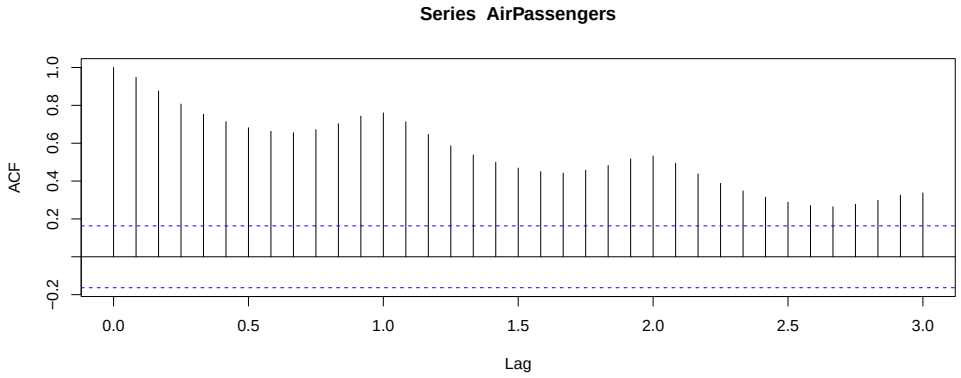
```
Call:
 stl(x = AirPassengers, s.window = "periodic")

 Time.series components:
    seasonal             trend            remainder
 Min.   :-57.48185   Min.   :126.1117   Min.   :-49.42378
 1st Qu.:-27.05842   1st Qu.:184.9487   1st Qu.:-10.30070
 Median : -7.01817   Median :260.2407   Median :  0.64024
 Mean   :  0.00000   Mean   :280.4517   Mean   : -0.15312
 3rd Qu.: 21.16290   3rd Qu.:373.3595   3rd Qu.: 10.84274
 Max.   : 70.24388   Max.   :497.4299   Max.   : 72.50607
 IQR:
     STL.seasonal STL.trend STL.remainder data
      48.22        188.41     21.14         180.50
   %  26.7         104.4      11.7          100.0

 Weights: all == 1

 Other components: List of 5
 $ win  : Named num [1:3] 1441 19 13
 $ deg  : Named int [1:3] 0 1 1
 $ jump : Named num [1:3] 145 2 2
 $ inner: int 2
 $ outer: int 0
```

```
plot(d)
```



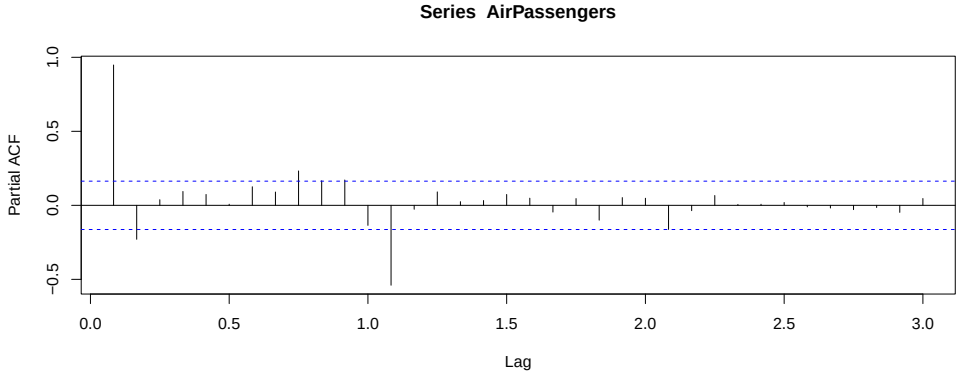

Автокорреляционная функция для исходных данных:

```
acf(AirPassengers,lag.max=36)
```



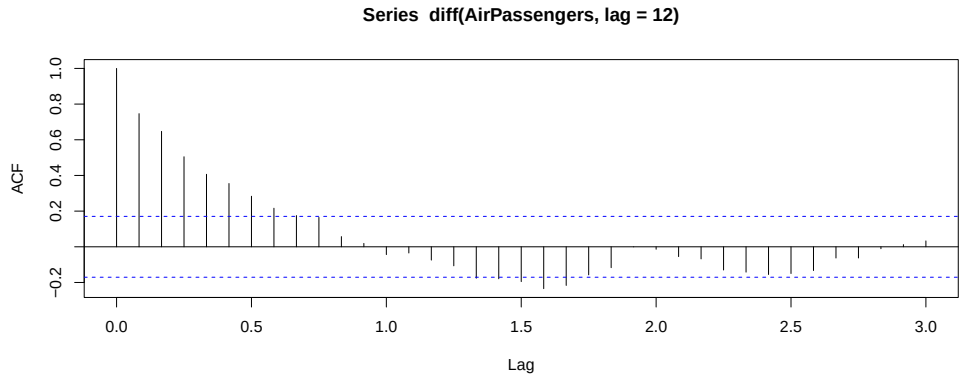

и частная автокорреляционная функция:

```
pacf(AirPassengers,lag.max=36)
```



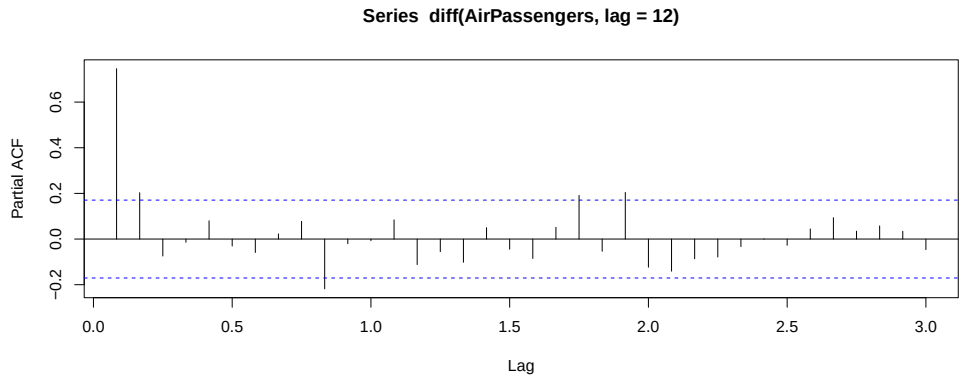
Автокорреляционная функция для $\delta_{12}x$:

```
acf(diff(AirPassengers,lag=12),lag.max=36)
```




и частная автокорреляционная функция

```
pacf(diff(AirPassengers,lag=12),lag.max=36)
```




SARIMA$(1,0,0)(1,1,0)_{12}$:

```
arima(AirPassengers,order=c(1,0,0),seasonal=list(order=c(1,1,0)))

Coefficients:
         ar1     sar1
      0.9513  -0.1479
s.e.  0.0256   0.1022

sigma^2 estimated as 143.9:  log likelihood = -516.48,  aic = 1038.95
```

SARIMA$(2,1,0)(1,1,0)_{12}$:

```
arima(AirPassengers,order=c(2,1,0),seasonal=list(order=c(1,1,0)))

Coefficients:
          ar1      ar2     sar1
      -0.3002  -0.0126  -0.1409
s.e.   0.0874   0.0887   0.0985

sigma^2 estimated as 134.7:  log likelihood = -507.19,  aic = 1022.37
```

Для сравнения ARIMA$(2,1,0)$:

```
arima(AirPassengers,order=c(2,1,0))
```

```
Coefficients:
         ar1      ar2
      0.3815  -0.2279
s.e.  0.0824   0.0834

sigma^2 estimated as 977.6:  log likelihood = -695.29,  aic = 1396.59
```

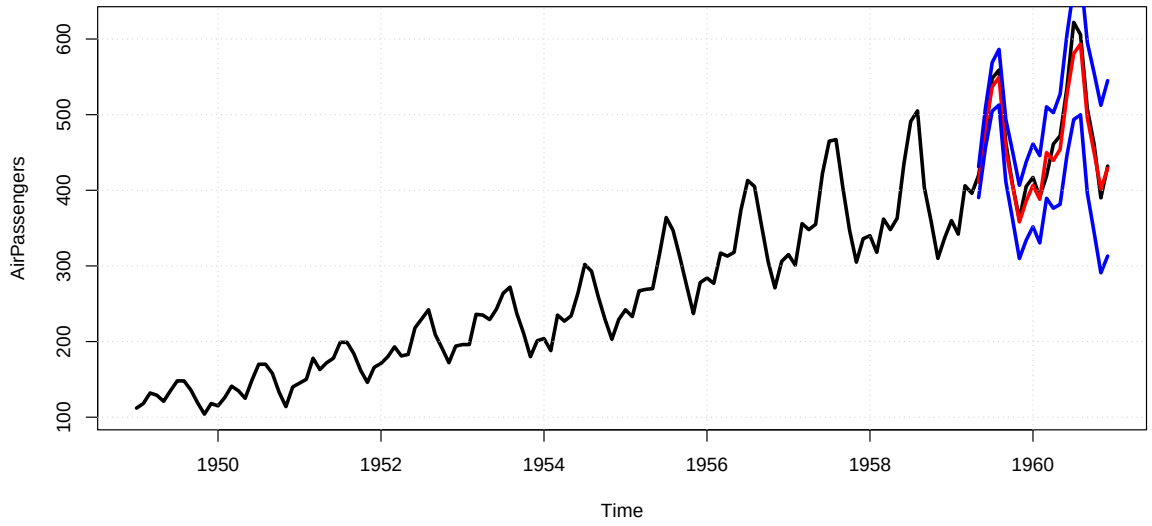
**Прогнозирование** Выделим $n_1$ наблюдений в конце и построим для них прогноз, используя модель, оценённую по $n_2 = n - n_1$ начальным значениям временного ряда.

```
n <- length(AirPassengers)
n1 <- 20      # количество прогнозных значений
n2 <- n-n1    # данные для настройки модели

X <- ts(AirPassengers[1:n2],
        start=start(AirPassengers),frequency=frequency(AirPassengers))
f <- arima(X,order=c(2,1,0),seasonal=list(order=c(1,1,0)))
p <- predict(f,n.ahead=n1); p
```

```
$pred
          Jan      Feb      Mar      Apr      May      Jun      Jul      Aug
1959                                    410.7485 482.2993 536.8818 549.5549
1960 406.5504 388.1072 449.8912 439.4480 454.2244 525.8249 580.5645 593.3846
          Sep      Oct      Nov      Dec
1959 452.7664 406.4368 358.2124 385.6556
1960 496.1295 449.9472 401.6369 429.0310
$se
          Jan      Feb      Mar      Apr      May      Jun      Jul      Aug
1959                                    10.26665 13.22901 16.22397 18.58537
1960 27.67260 29.15048 30.55706 31.90165 36.77443 40.34032 43.97881 47.23208
          Sep      Oct      Nov      Dec
1959 20.73681 22.66631 24.45054 26.11090
1960 50.31544 53.20700 55.95414 58.57090
```

График результатов:

```
plot(AirPassengers,lwd=3)                 # истинные значения
lines(p$pred,lwd=3,col="red")             # прогноз

t <- qt(0.975,n2-2)
lines(p$pred+p$se*t,lwd=3,col="blue")     # доверительная воронка (95%)
lines(p$pred-p$se*t,lwd=3,col="blue")

grid()                                    # добавим сетку
```




### 5.3.2 Модель авторегрессионной условной гетероскедастичности

Вид модели GARCH$(m, s)$:

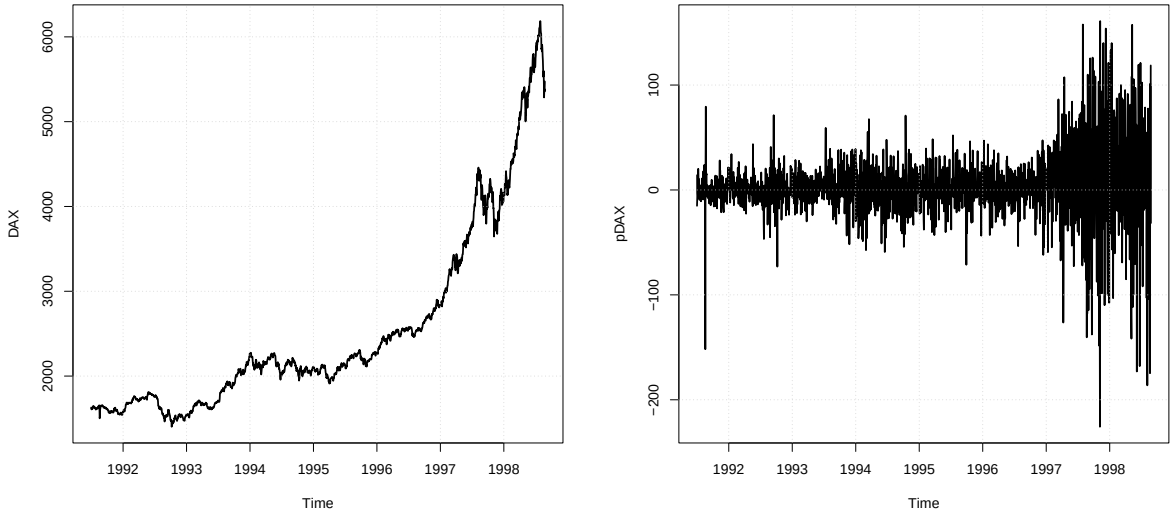
$$y_t = \mu_t + \varepsilon_t$$

$$\varepsilon_t = \sigma_t \nu_t, \quad \sigma_t^2 = \alpha_0 + \sum_{i=1}^{m} \alpha_i \varepsilon_{t-i}^2 + \sum_{j=1}^{s} \beta_j \sigma_{t-j}^2, \quad \nu_t \sim iid(0,1).$$

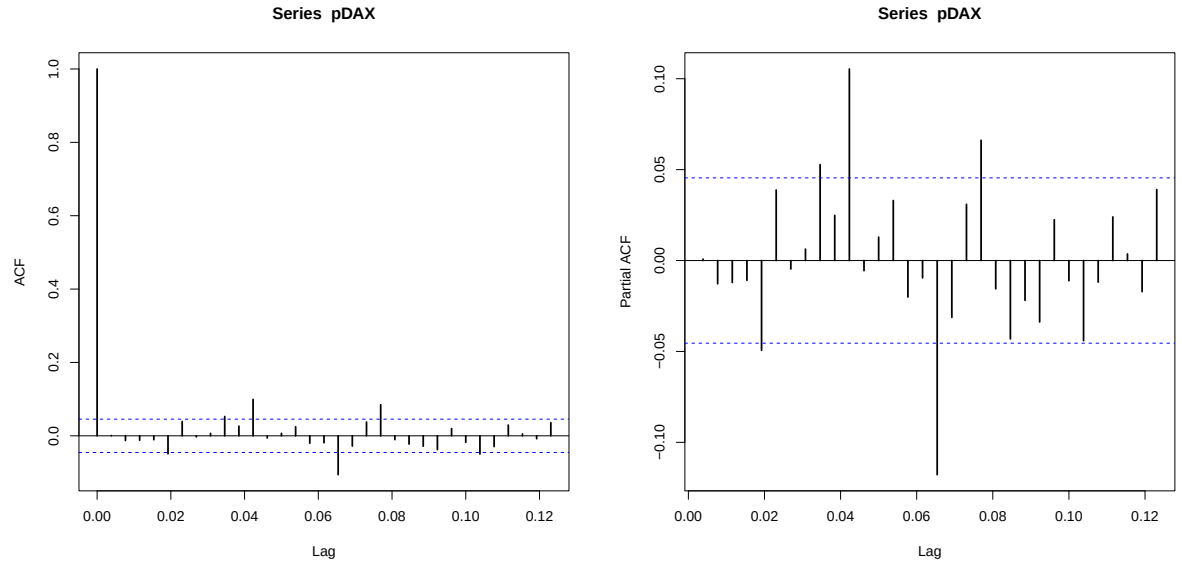
Для работы с обобщёнными авторегрессионными моделями с условной гетероскедастичностью (GARCH) используется функция `garch` из библиотеки `tseries` (анализ временных рядов и вычислительные финансы).

**Пример.** *Дневные цены закрытия DAX в 1991–1998 гг.*

```
DAX <- EuStockMarkets[,1]
plot(DAX,lwd=2); grid()

pDAX <- diff(DAX)
plot(pDAX,lwd=2); grid()
```



```
acf(pDAX,lwd=2)
pacf(pDAX,lwd=2)
```




```
library(tseries)
garch(pDAX,order=c(1,1),grad='numerical',trace=F) -> pDAX.arch
summary(pDAX.arch)
```

```
Call:
garch(x = pDAX, order = c(1, 1), grad = "numerical", trace = F)
Model:
GARCH(1,1)
Residuals:
     Min       1Q   Median       3Q      Max
-6.36822 -0.48320  0.04839  0.68180  4.79661
Coefficient(s):
    Estimate  Std. Error  t value Pr(>|t|)
a0  1.449280    0.429698    3.373 0.000744 ***
a1  0.039105    0.001980   19.745  < 2e-16 ***
b1  0.960138    0.001484  647.125  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Diagnostic Tests:
        Jarque Bera Test
data:  Residuals
X-squared = 368.6017, df = 2, p-value < 2.2e-16
        Box-Ljung test
data:  Squared.Residuals
X-squared = 1.1213, df = 1, p-value = 0.2896
```

# Глава 6

# Data Mining

Описание методов Data Mining применительно к R можно посмотреть, например, в [25] и [26].

## 6.1 Регрессия

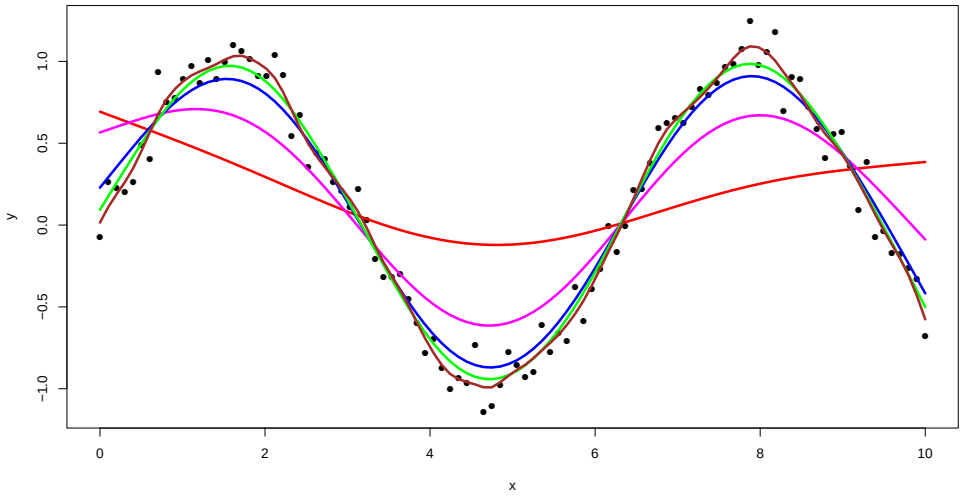
### 6.1.1 Сглаживающие сплайны

Использование сглаживающих сплайнов позволяет подобрать гладкую функцию, аппроксимирующую заданные значения (чем больше значение `df`, тем более точно функция описывает данные).

**Пример.** *Аппроксимация*

$$y = \sin(x) + \varepsilon, \qquad 0 \leqslant x \leqslant 10, \quad \varepsilon \sim \mathcal{N}(0, 0.1^2).$$

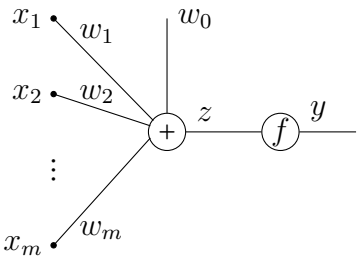
```
x <- seq(from=0,to=10,length=100)
y <- sin(x)+rnorm(length(x),0,0.1)

DF <- c(3,5,7,9,25)
COL <- c('red','magenta','blue','green','brown')

plot(x,y,pch=16)
for (i in 1:length(DF))
  lines(smooth.spline(x,y,df=DF[i]),col=COL[i],lwd=3)
```



### 6.1.2 Многослойный персептрон

Подробное описание многослойного персептрона и сети на основе радиально-базисных функций см. [19]. Описание рекуррентных и свёрточных сетей — [17]. Использование методов глубокого обучения в R описывается в [24].

Искусственный нейрон-персептрон имеет вид

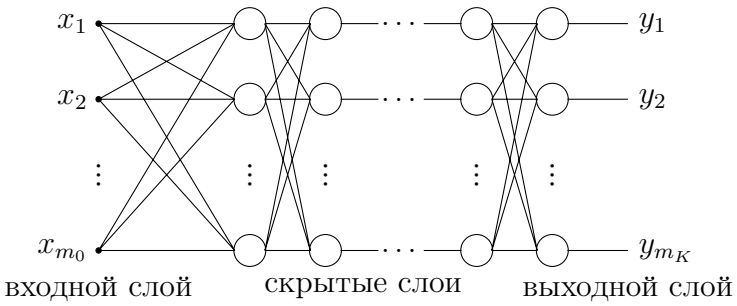



- $x_i$ — входные значения.
- $w_i$, $i = 1, \ldots, n$ — веса входов.
- $w_0$ — значение сдвига для реализации аффинных функций.
- $f$ — функция активации.
  - Единичная ступенька (функция Хевисайда):
$$f(z) = \begin{cases} 0, & z < 0, \\ 1, & z \geqslant 0. \end{cases}$$
  - Линейная функция активации: $f(z) = z$.
  - Логистическая функция активации (униполярный сигмоид):
$$f(z) = \frac{1}{1 + \exp(-z)}.$$
  - ReLU (Rectified Linear Unit, (полу)линейный выпрямитель, полулинейный элемент):
$$f(z) = \begin{cases} 0, & z < 0, \\ z, & z \geqslant 0. \end{cases}$$

При использовании линейной функции активации, искусственный нейрон реализует модель линейной регрессии

$$\hat{y} = w_0 + \sum_{j=1}^{m} w_j x_j.$$

При объединении нескольких нейронов в слой, а нескольких слоёв в сеть, получается полносвязная сеть прямого распространения:




В нейронах скрытых слоёв используются нелинейные функции активации. Если в нейронах выходного слоя применяются линейные функции активации, то сеть решает задачу построения моделей регрессии. Если в выходном слое функции активации — единичная ступенька или логистическая, то сеть решает задачу (множественной) классификации.

В принципе, уже трёхслойная сеть (с одним скрытым слоем) является универсальным аппроксиматором. Введение дополнительных скрытых слоёв позволяет уменьшить количество нейронов (в каждом слое и общее), при сохранении аппроксимирующих возможностей сети.

Библиотека `nnet` в `R` содержит функцию `nnet`, предназначенную для построения и обучения трёхслойных нейронных сетей с линейной или логистической (по выбору) функцией на выходе.

```
nnet(x, ...)

## S3 method for class 'formula'
nnet(formula, data, weights, ...,
     subset, na.action, contrasts = NULL)

## Default S3 method:
nnet(x, y, weights, size, Wts, mask,
     linout = FALSE, entropy = FALSE, softmax = FALSE,
     censored = FALSE, skip = FALSE, rang = 0.7, decay = 0,
     maxit = 100, Hess = FALSE, trace = TRUE, MaxNWts = 1000,
     abstol = 1.0e-4, reltol = 1.0e-8, ...)
```

## 6.2 Классификация

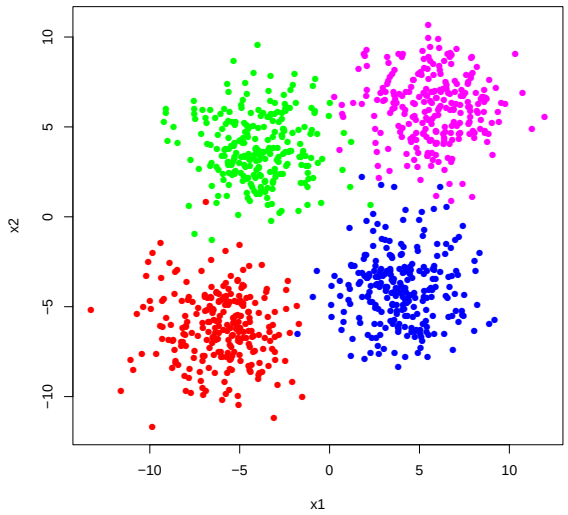
### 6.2.1 Модель множественного выбора

Пусть имеется обучающее множество объектов $x_i \in \mathbb{R}^m$, причём каждый объект имеет маркер $y_i \in \mathsf{C} \subset \mathbb{N}$ ($\operatorname{mes}\mathsf{C} = M \geqslant 2$): $\{(x_i, y_i)\}$, $i = 1, \ldots, n$. Требуется построить классификатор, относящий произвольный объект $x \in \mathbb{R}^m$ к определённому классу $\mathsf{C}$.

Если $M = 2$, то можно использовать модель бинарного выбора (см. стр. 152).

```
# обучающее множество
n <- 250

s <- 2
x1 <- c(rnorm(n,mean=-5,sd=s),  rnorm(n,mean=-5,sd=s),
        rnorm(n,mean=5,sd=s), rnorm(n,mean=5,sd=s))
x2 <- c(rnorm(n,mean=-5,sd=s),  rnorm(n,mean=5,sd=s),
        rnorm(n,mean=-5,sd=s), rnorm(n,mean=5,sd=s))

# четыре класса
```

```
N <- 4
y <- c(rep(1,n),rep(2,n),rep(3,n),rep(4,n))

# цвета для точек
COLOR <- c(rgb(1,0,0),rgb(0,1,0),rgb(0,0,1),rgb(1,0,1))
# цвета для фона
color <- c(rgb(1,.9,.9),rgb(.9,1,.9),rgb(.9,.9,1),rgb(1,.9,1))

plot(x1,x2,type='n')
for (i in 1:N) points(x1[y==i],x2[y==i],pch=16,col=COLOR[i])
```




При $M > 2$ для каждого класса строится отдельный бинарный классификатор, отделяющий его от всех остальных классов. При этом все классификаторы настраиваются одновременно, чтобы минимизировать общую суммарную ошибку.

При построении модели бинарного выбора предполагается, что $y \in \{0, 1\}$. Введём для каждого класса $\mathsf{C}_j$ «фиктивные» маркеры

$$y_i^j = \begin{cases} 1, & y_i = \mathsf{C}_j, \\ 0, & y_i \neq \mathsf{C}_j. \end{cases}$$

```
# матрица «фиктивных» маркеров
Y <- array(0,dim=c(length(x1),N))
for (i in 1:N) Y[y==i,i] <- 1
```

Вид матрицы $X$ определяет вид дискриминантной функции. Например, при $X = (1, x_1, x_2)$ дискриминантная функция — линейная:

$$y^{\star,j} = \beta_0^j + \beta_1^j x_1 + \beta_1^j x_2.$$

А если $X = (1, x_1, x_2, x_1^2, x_1 x_2, x_2^2)$, то — квадратичная:

$$y^{\star,j} = \beta_0^j + \beta_1^j x_1 + \beta_1^j x_2 + \beta_3^j x_1^2 + \beta_4^j x_1 x_2 + \beta_5^j x_2^2.$$

```
RMk <- function(x1,x2) c(1,x1,x2)
X <- c()
for (i in 1:length(x1)) X <- rbind(X,RMk(x1[i],x2[i]))
```

Объединив все $\beta_i^j$ в матрицу $B = \{\beta_i^j\}$ можно вычислять значения всех дискриминантных функций одновременно:

$$Y^\star = XB.$$

Столбцы матрицы $Y^\star$ нормируются: $Y^\star_{\cdot j} \leftarrow Y^\star_{\cdot j}/s_{Y^\star_{\cdot j}}$, чтобы их среднеквадратическое отклонение было равно единице. После этого для всех элементов матрицы $Y^\star$ выполняется логистическое преобразование:

$$\Phi_{ij} = \frac{1}{1+\exp(-Y^\star_{ij})}.$$

Аналогично модели бинарного выбора, будем считать, что

$$\Phi_{ij} = \mathsf{P}\{y_i = \mathsf{C}_j\}.$$

Тогда обучение модели множественного выбора сводится к подбору таких коэффициентов $B$, чтобы

$$\sum_{i=1}^{n}\sum_{j\in\mathsf{C}}(y_i^j - \Phi_{ij})^2 \to \min.$$

```
# метод наименьших квадратов
MS <- function(b,X,Y)
{
  B <- array(dim=c(dim(X)[2],N))
  for (i in 1:N) B[,i] <- b[((i-1)*dim(X)[2]+1):(i*dim(X)[2])]

  Ys <- X%*%B
  for (i in 1:N) Ys[,i] <- Ys[,i]/sd(Ys[,i])
  Phi <- 1/(1+exp(-Ys))
  sum((Y-Phi)^2)
}
```

Так как возможно попадание при поиске в точку локального минимума, то обучение модели выполняется несколько раз, начиная со случайного набора значений $B$. В качестве результата выбирается модель с минимальной суммарной ошибкой.

```
optim(runif(dim(X)[2]*N,min=-10,max=10),
      function(b) MS(b,X,Y), method='BFGS') -> res
for (i in 1:10)
{
  optim(runif(dim(X)[2]*N,min=-10,max=10),
        function(b) MS(b,X,Y), method='BFGS') -> tmp
  if (tmp$value < res$value) res <- tmp
}
Brow <- res$par

B <- array(dim=c(dim(X)[2],N))
for (i in 1:N) B[,i] <- Brow[((i-1)*dim(X)[2]+1):(i*dim(X)[2])]
```

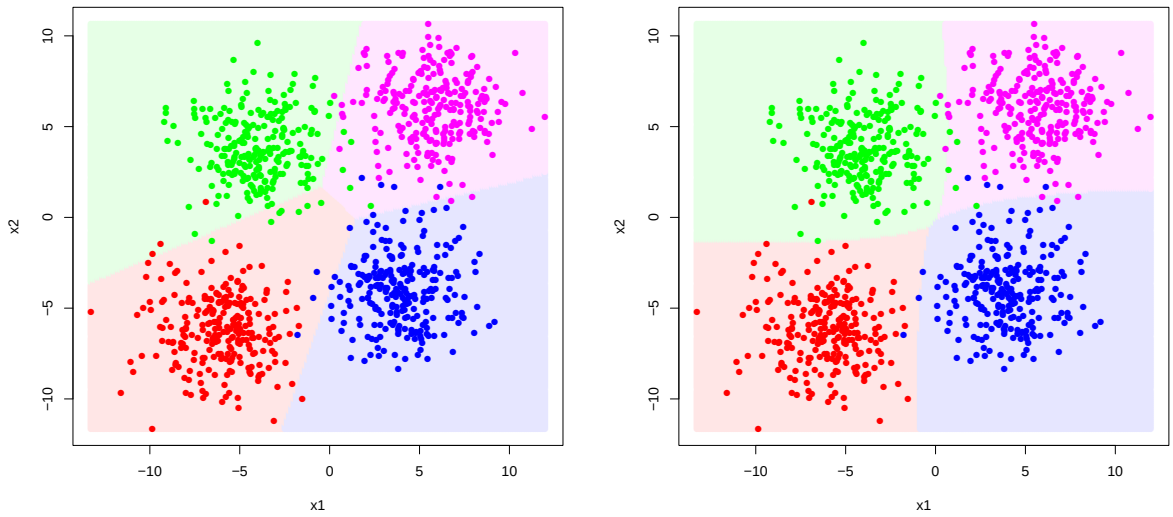
При визуализации подмножества $\mathbb{R}^2$, относящиеся к разным классам, закрасим бледными цветами, на которые наложим исходные данные.

```
# визуализация результата
plot(x1,x2,type='n')
T1 <- seq(from=min(x1),to=max(x1),length=200)
T2 <- seq(from=min(x2),to=max(x2),length=200)
for (t1 in T1) for (t2 in T2)
```

```
{
  ym <- RMk(t1,t2)%*%B
  points(t1,t2,pch=16,col=color[which(ym==max(ym))])
}
for (i in 1:N) points(x1[y==i],x2[y==i],pch=16,col=COLOR[i])
```

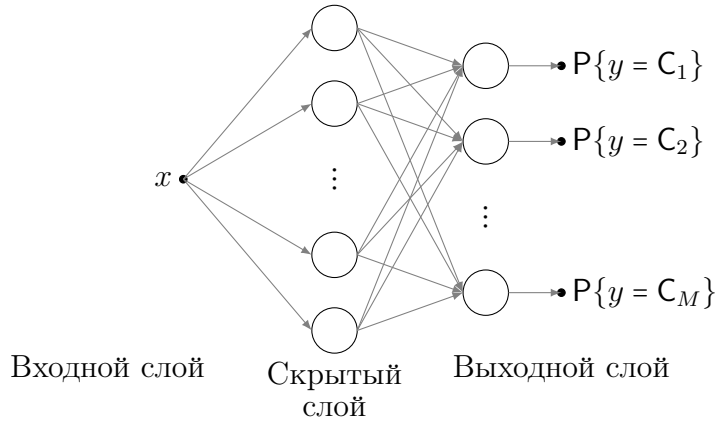
На левом рисунке показаны области, получаемые с помощью линейных классификаторов, на правом — квадратичных.

```
RMk <- function(x1,x2) c(1,x1,x2,x1^2,x1*x2,x2^2)
```




### 6.2.2 Многослойный персептрон

Похожего результата можно достичь, используя в качестве классификатора многослойный персептрон вида

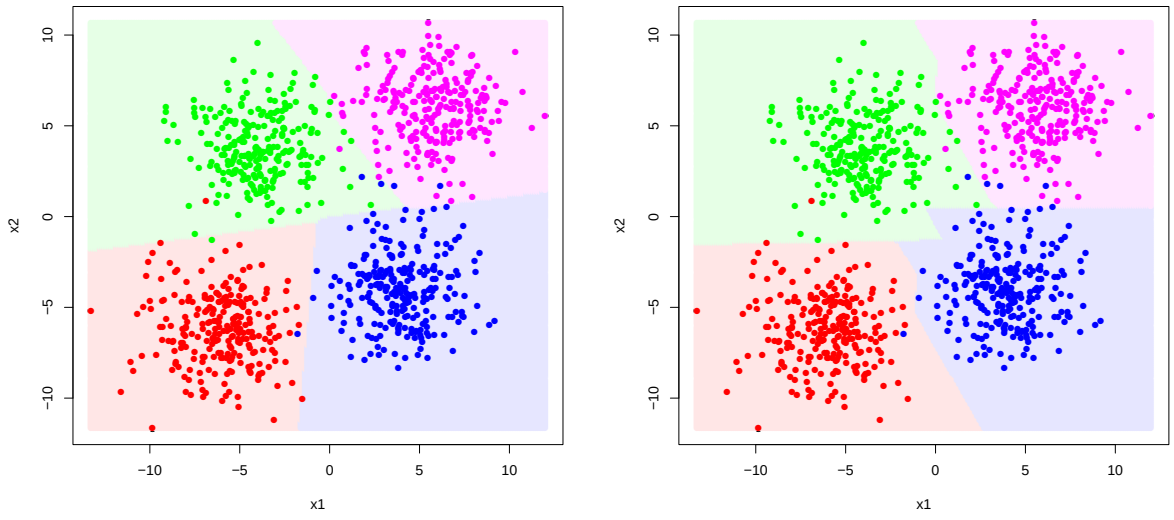



Функция `nnet` из библиотеки `nnet` создаёт и обучает такой трёхслойный персептрон (см. стр. 174).

```
# для нейронной сети всегда матрицу X составляют только значения на входе X = (x1, x2)
X <- cbind(x1,x2)

library(nnet)
```

```
SZ <- 3                 # количество нейронов в скрытом слое

nnet(X,Y,size=SZ) -> net
for (i in 1:10)         # несколько циклов обучения, чтобы снизить
{                       # вероятность попасть в локальный минимум
  nnet(X,Y,size=SZ) -> tmp
  if (tmp$value < net$value) net <- tmp
}

# визуализация результата
plot(x1,x2)
T1 <- seq(from=min(x1),to=max(x1),length=200)
T2 <- seq(from=min(x2),to=max(x2),length=200)
for (t1 in T1) for (t2 in T2)
{
  ym <- predict(net,c(t1,t2))
  points(t1,t2,pch=16,col=color[which(ym==max(ym))])
}
for (i in 1:N) points(x1[y==i],x2[y==i],pch=16,col=COLOR[i])
```

Результаты классификации для сетей с тремя (слева) и пятью (справа) нейронами в скрытом слое.

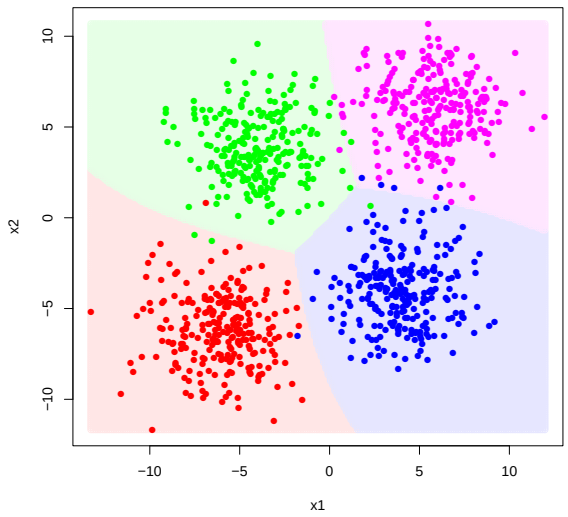



### 6.2.3 Машина опорных векторов

Функция svm из библиотеки e1071 создаёт и настраивает машину опорных векторов (см. [19: Гл. 6]).

```
## S3 method for class 'formula'
svm(formula, data = NULL, ..., subset, na.action = na.omit, scale = TRUE)
## Default S3 method:
svm(x, y = NULL, scale = TRUE, type = NULL,
    kernel = "radial", degree = 3,
    gamma = if (is.vector(x)) 1 else 1 / ncol(x),
    coef0 = 0, cost = 1, nu = 0.5,
    class.weights = NULL, cachesize = 40,
```

```
    tolerance = 0.001, epsilon = 0.1,
    shrinking = TRUE, cross = 0, probability = FALSE, fitted = TRUE,
    ..., subset, na.action = na.omit)

library(e1071)

X <- cbind(x1,x2)
Y <- as.factor(y)

svm(X,Y,probability=TRUE) -> f

# визуализация результата
TT <- c()
for (i in 1:length(T1))
for (j in 1:length(T2)) TT <- rbind(TT,c(T1[i],T2[j]))
predict(f,TT) -> YY

plot(x1,x2,type='n')
for (i in 1:N) points(TT[YY==i,1],TT[YY==i,2],col=color[i])
for (i in 1:N) points(x1[y==i],x2[y==i],pch=16,col=COLOR[i])
```



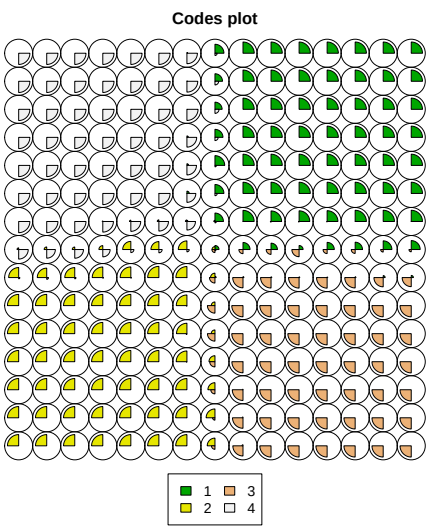

### 6.2.4 Контекстные карты

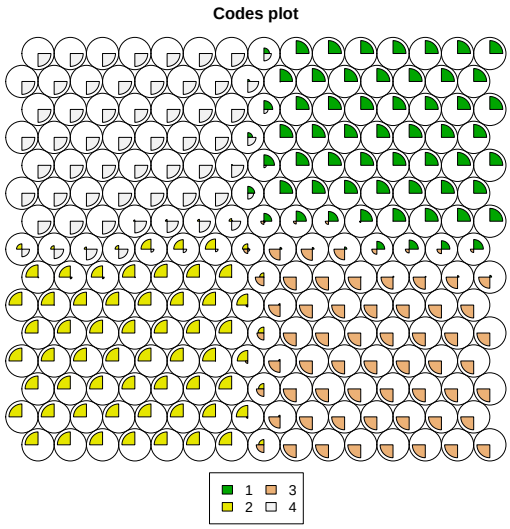
Функция `xyf` в библиотеке `kohonen` используется для построения контекстных карт на основе самоорганизующихся сетей (см. [19: Гл. 9]).

```
xyf(X, Y, ...)
```

Вид самоорганизующейся сети задаётся функцией `somgrid`.

```
somgrid(xdim = 8, ydim = 6, topo = c("rectangular", "hexagonal"),
        neighbourhood.fct = c("bubble", "gaussian"), toroidal = FALSE)

library(kohonen)
plot(xyf(X,Y,grid=somgrid(15,15,'rectangular')))
plot(xyf(X,Y,grid=somgrid(15,15,'hexagonal')))
```



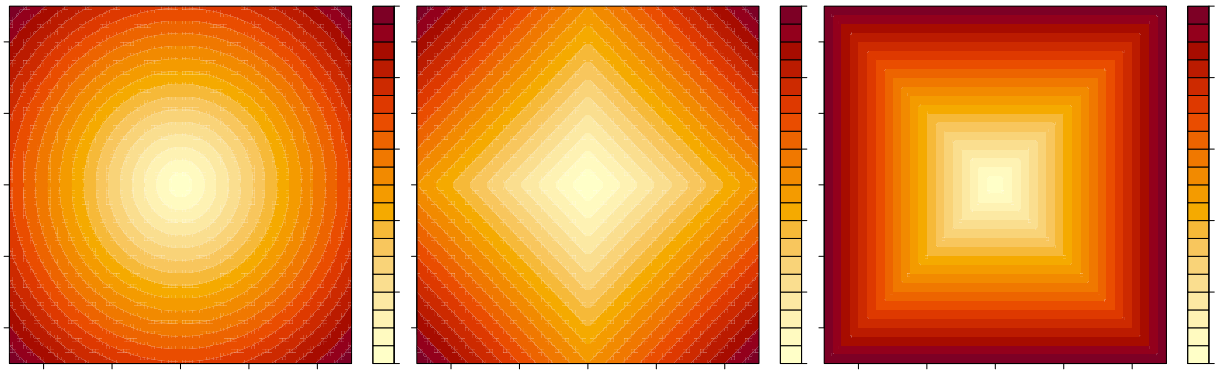




## 6.3 Кластерный анализ

Расстояния Евклида, Минковского (манхэттенское, городских кварталов) и Чебышёва между $x, y \in \mathbb{R}^m$:

```
dist <- function(x,y,type='euclidean')
{
  if (type=='manhattan') return(sum(abs(x-y)))
  if (type=='chebyshev') return(max(abs(x-y)))
  sum((x-y)^2)^0.5
}
```

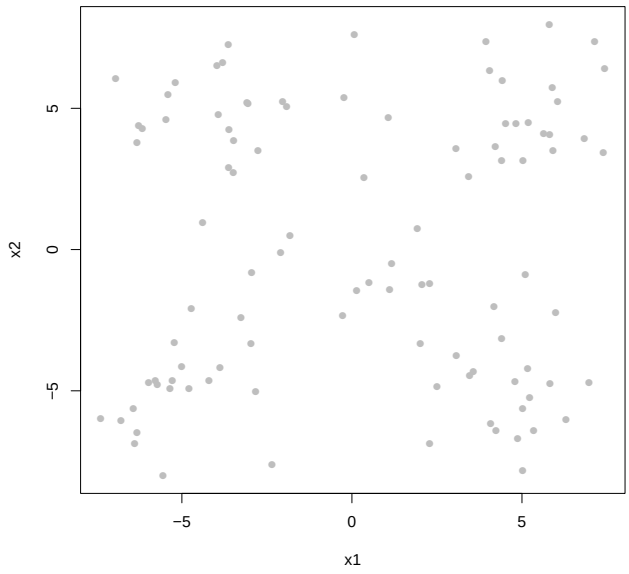
Линии уровня (карты высот) расстояний от начала координат в метриках Евклида (слева), Минковского (в середине) и Чебышёва (справа).



Исходные данные.

```
n <- 20

s <- 1.5
x1 <- c(rnorm(n,mean=-5,sd=s),rnorm(n,mean=5,sd=s),rnorm(n,mean=0,sd=s*1.5),
        rnorm(n,mean=-5,sd=s),rnorm(n,mean=5,sd=s))
x2 <- c(rnorm(n,mean=-5,sd=s),rnorm(n,mean=5,sd=s),rnorm(n,mean=0,sd=s*1.5),
        rnorm(n,mean=5,sd=s),rnorm(n,mean=-5,sd=s))
n <- length(x1)
```

```
X <- cbind(x1,x2)
plot(x1,x2,pch=16,col='gray')
```




### 6.3.1 Иерархические алгоритмы

#### Агломеративные алгоритмы

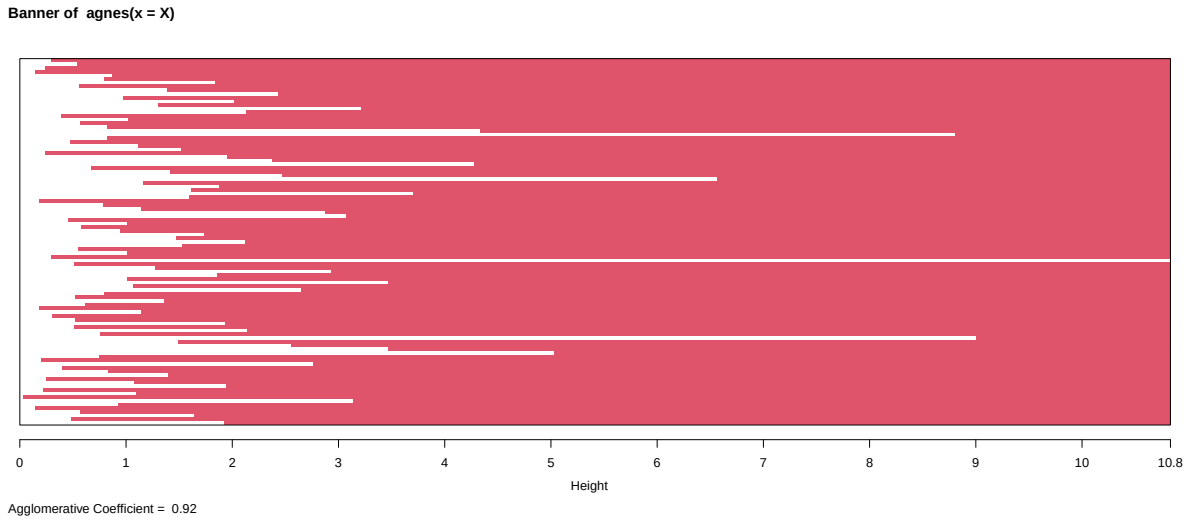
Начиная с кластеров-синглетонов[1] постепенное объединение до одного кластера. Функция agnes из библиотеки cluster:

```
agnes(x, diss = inherits(x, "dist"), metric = "euclidean",
      stand = FALSE, method = "average", par.method,
      keep.diss = n < 100, keep.data = !diss, trace.lev = 0)
```

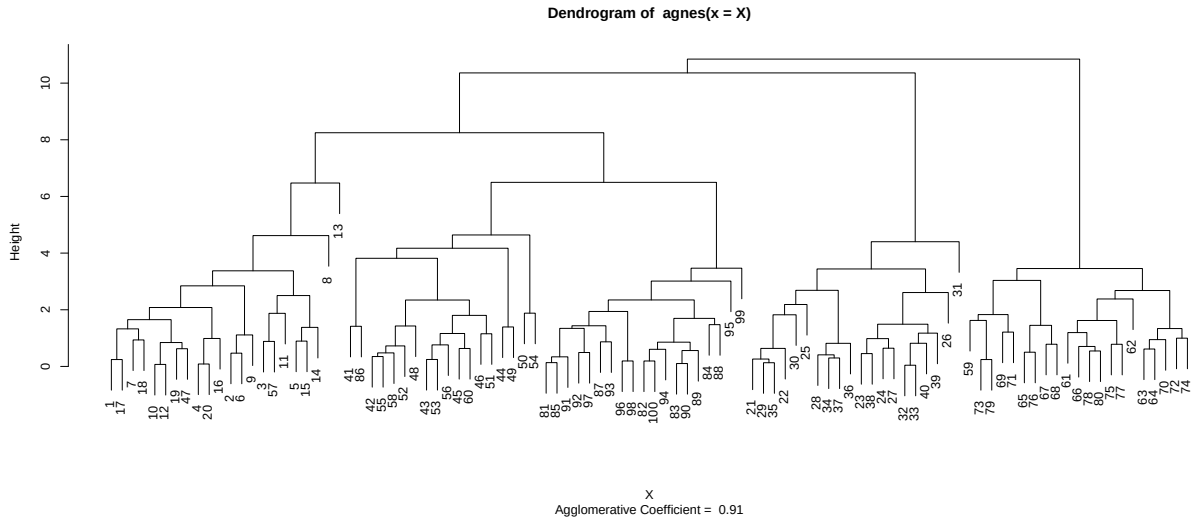
```
library(cluster)
```

```
agnes(X) -> res; res
plot(res,which=1)
```

[1] Синглетон — множество, содержащее только один элемент.




```
plot(res,which=2)
```

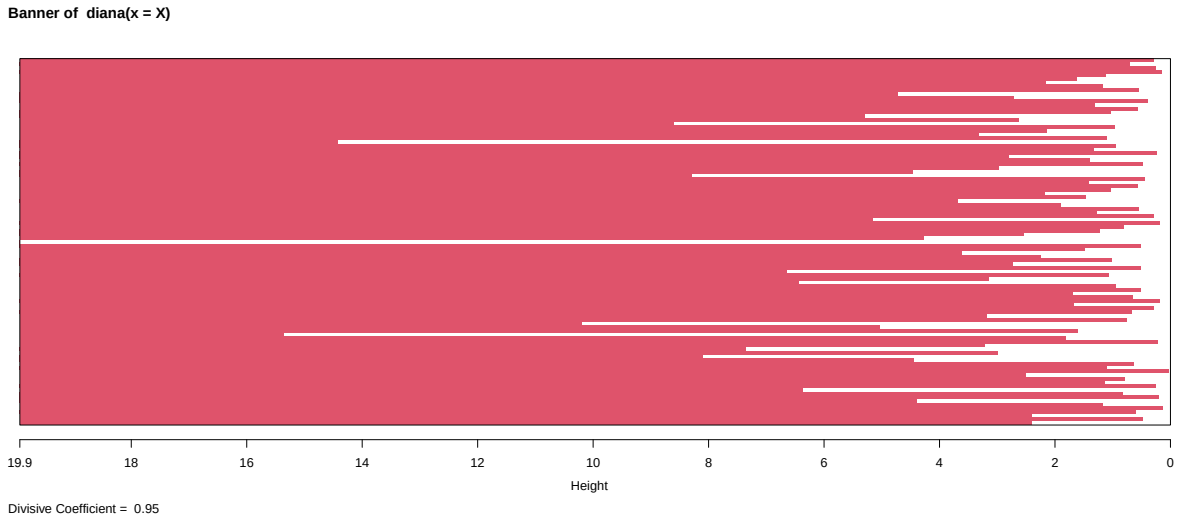



### Дивизимные алгоритмы

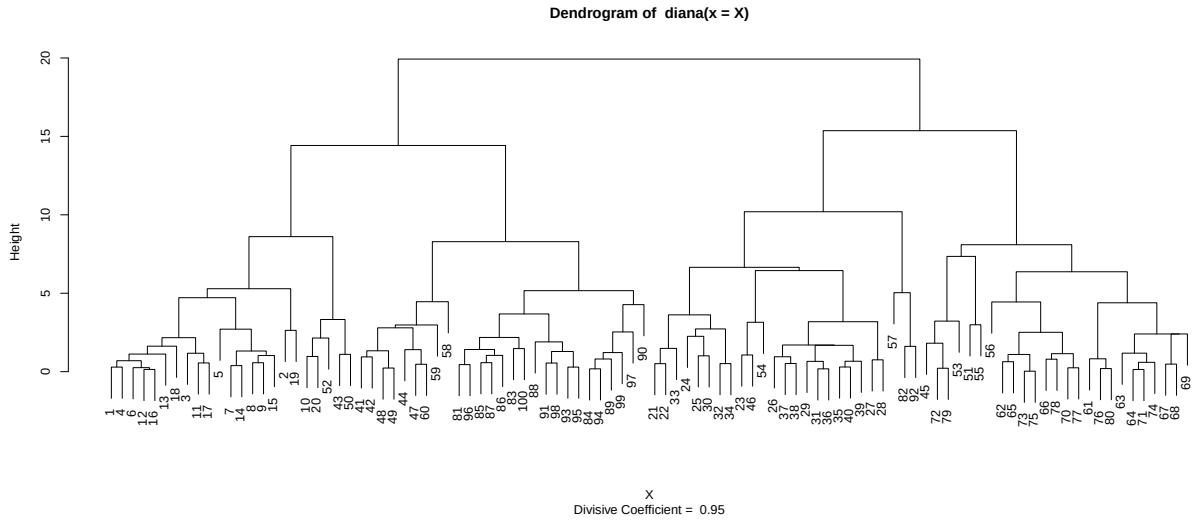
Начиная с одного кластера, представляюего всё исходное множество, постепенное разбиение на подмножества, пока не останутся только кластеры-синглетоны. Функция `diana` из библиотеки `cluster`:

```
diana(x, diss = inherits(x, "dist"), metric = "euclidean", stand = FALSE,
      stop.at.k = FALSE,
      keep.diss = n < 100, keep.data = !diss, trace.lev = 0)
```

```
diana(X) -> res; res
plot(res,which=1)
```




```
plot(res,which=2)
```




### 6.3.2 Неиерархические алгоритмы

#### Метод $k$ средних

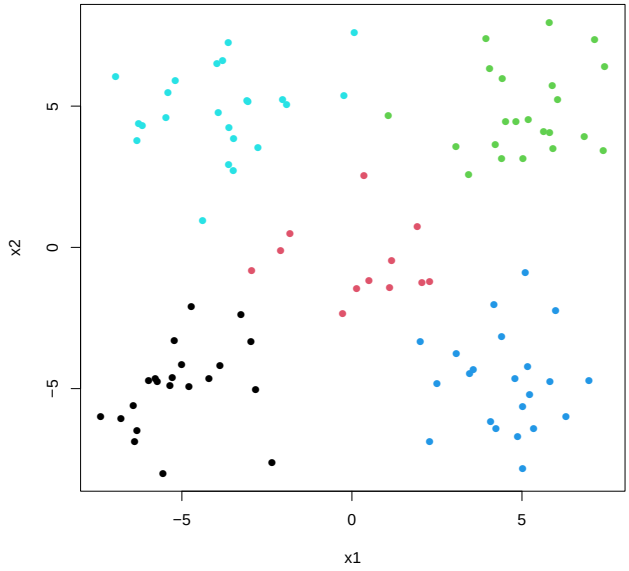
Метод $k$ средних применяется при разбиении исходного множества на $k$ кластеров. Алгоритм метода достаточно прост.

1. Определить $k$ центров кластеров $\{C_1, \ldots, C_k\}$.
2. Разбить множество $\{x_i\}_1^n$ на кластеры. Точка $x_i$ относится к тому кластеру, чей центр являтся ближайшим:
$$c_i = \arg\min_j \operatorname{dist}(x_i, C_j).$$
3. Переопределить центры кластеров:
$$C_j = \frac{\sum_{i=1}^n \delta_{c_i,j} x_i}{\sum_{i=1}^n \delta_{c_i,j}}.$$
4. Повторять второй и третий шаги до тех пор, пока кластеры не перестанут меняться.

```
K <- 5                          # количество кластеров
CL <- X[1:K,]                   # исходные центры кластеров
color <- rainbow(K)

cl <- integer(n)                # классы точек
cl_prev <- cl

repeat
{
  # точка принадлежит кластеру с ближайшим центром
  for (i in 1:n)
  {
    d <- sapply(1:K,function(j) dist(X[i,],CL[j,]))
    cl[i] <- which.min(d)
  }
  # пересчёт центров
  for (i in 1:K) CL[i,] <- colMeans(X[cl==i,])

  # проверка на изменения кластеров
  if (all(cl==cl_prev)) break
  cl_prev <- cl
}

# холст
plot(x1,x2,pch=16,type='n')
# раскрашенные кластеры
for (i in 1:K) points(x1[cl==i],x2[cl==i],pch=16,col=color[i])
# центры кластеров
points(CL[,1],CL[,2],pch=17)
```

Конечно, можно также использовать функцию kmeans из библиотеки cluster.

```
kmeans(x, centers, iter.max = 10, nstart = 1,
       algorithm = c("Hartigan-Wong", "Lloyd", "Forgy",
                     "MacQueen"), trace=FALSE)

kmeans(X,5) -> cl
plot(X,pch=16,col=cl$cluster)
```

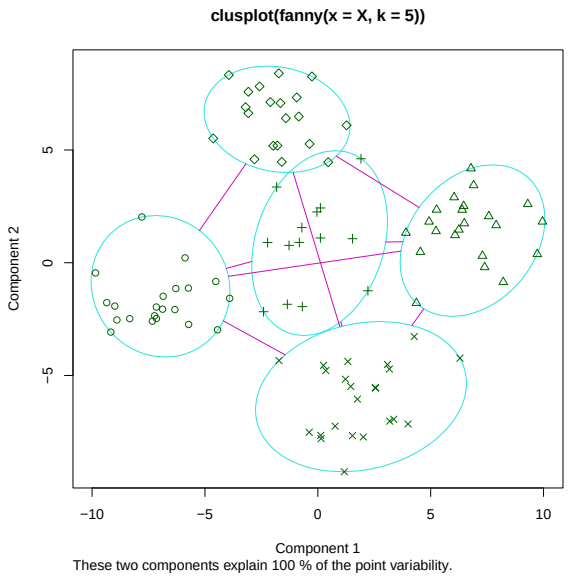


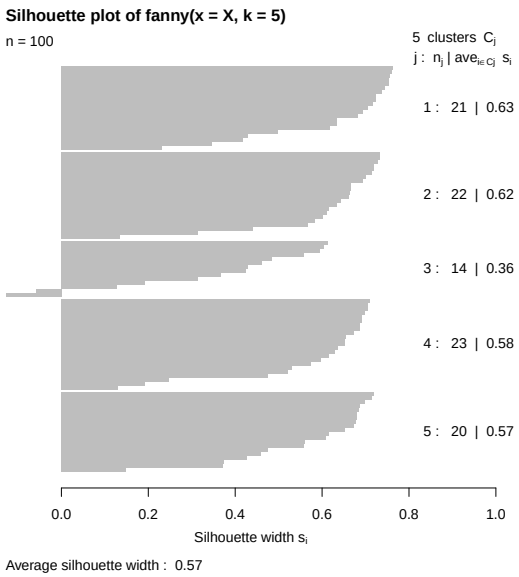

### Метод $k$ нечётких средних

Метод $k$ нечётких средних разбивает исходное множество на кластеры, каждый из которых представлен в виде нечёткого множества. Функция `fanny` из библиотеки `cluster`:

```
fanny(x, k, diss = inherits(x, "dist"), memb.exp = 2,
      metric = c("euclidean", "manhattan", "SqEuclidean"),
      stand = FALSE, iniMem.p = NULL, cluster.only = FALSE,
      keep.diss = !diss && !cluster.only && n < 100,
      keep.data = !diss && !cluster.only,
      maxit = 500, tol = 1e-15, trace.lev = 0)
```

```
fanny(X,5) -> cl

plot(cl,which=1)
plot(cl,which=2)
```



### 6.3.3 Использование нечёткой $\alpha$-квазиэквивалентности

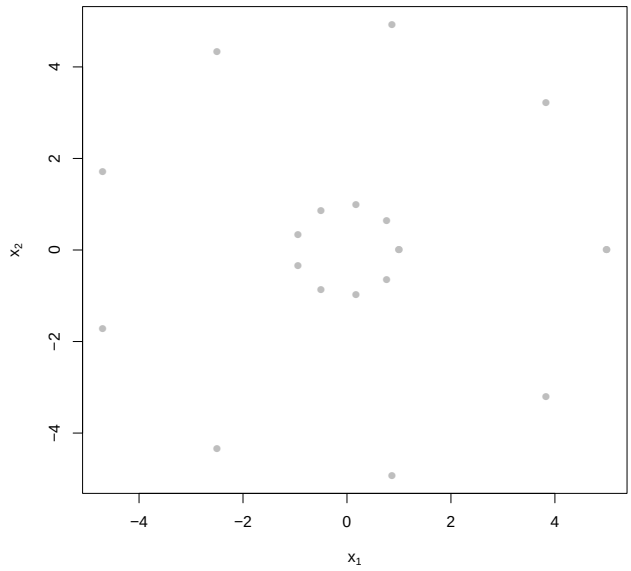
Исходные данные, на которых не будут работать алгоритмы кластеризации, основанные на расстояниях до центров:

```
n <- 10
t <- seq(from=0,to=2*pi,length=n)

S <- .1; s <- .1
R <- 5; r <- 1
x1 <- c(R*cos(t),r*cos(t))
x2 <- c(R*sin(t),r*sin(t))
X <- cbind(x1,x2)
N <- dim(X)[1]
```

Алгоритмы кластеризации типа $k$ средних определяют степень принадлежность объекта определённому кластеру, сравнивая расстояния от этого объекта до центров кластеров.

Однако в данном случае можно интуитивно разделить всё множество на точки, составляющие внешнюю и внутреннюю окружность, которые имеют при этом *один* центр!

```
plot(X[,1],X[,2],pch=16,col='gray',
     xlab=expression(x[1]),ylab=expression(x[2]))
```




В таком случае для разделения множества на кластеры можно использовать понятие нечёткой $\alpha$-квазиэквивалентности ([13: стр. 182–204]).

$d_{ij}$ — расстояние между $i$-й и $j$-й точками.

```
d <- array(0,dim=c(N,N))
for (i in 1:N)
  for (j in i:N)
  {
    d[i,j] <- dist(X[i,],X[j,])
    d[j,i] <- d[i,j]
  }
```

$m_{ij}$ — оценка сходства между $i$-й точкой и $j$-м образцом:

$$m_{ij} = 1 - \frac{d_{ij}}{\max_j d_{ij}}.$$

```
m <- array(0,dim=c(N,N))
for (i in 1:N)
  m[i,] <- 1-d[i,]/max(d[i,])
```

$R_{ij}$ — значение отношения сходства между $i$-й и $j$-й точками относительно *всех* образцов:

$$R_{ij} = 1 - \max_k |m_{ik} - m_{jk}|.$$

```
R <- array(0,dim=c(N,N))
for (i in 1:N)
  for (j in 1:N)
    R[i,j] <- 1-max(abs(m[i,]-m[j,]))
```

$R$ — отношение толерантности (рефлексивное и симметричное). $R^\star$ — транзитивное замыкание отношения $R$:

$$R^1_{ij} = R_{ij}, \quad R^{m+1}_{ij} = \max_k \min(R^m_{ik}, R_{kj}), \quad m = 1, 2, \ldots, n,$$
$$R^\star = \bigcup_k R^k,$$

является отношением эквивалентности.

```
Rm <- R
Rz <- R
for (ii in 1:N)
{
  tmp <- array(0,dim=c(N,N))
  for (i in 1:N)
    for (j in 1:N)
      for (k in 1:N) tmp[i,j] <- max(tmp[i,j],min(Rm[i,k],R[k,j]))
  Rm <- tmp

  for (i in 1:N)
    for (j in 1:N)
      Rz[i,j] <- max(Rz[i,j],Rm[i,j])
}
```

Пороговые значения (уникальные значения в матрице $R^\star$):

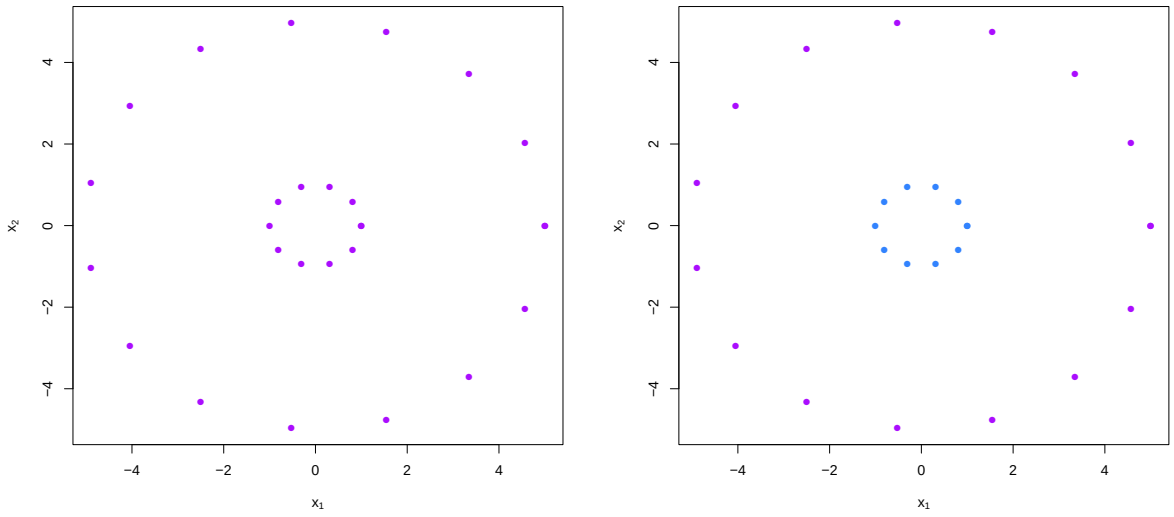
```
unique(sort(as.vector(round(Rz,digits=5))))
```

```
[1] 0.33130 0.79094 0.89437 1.00000
```

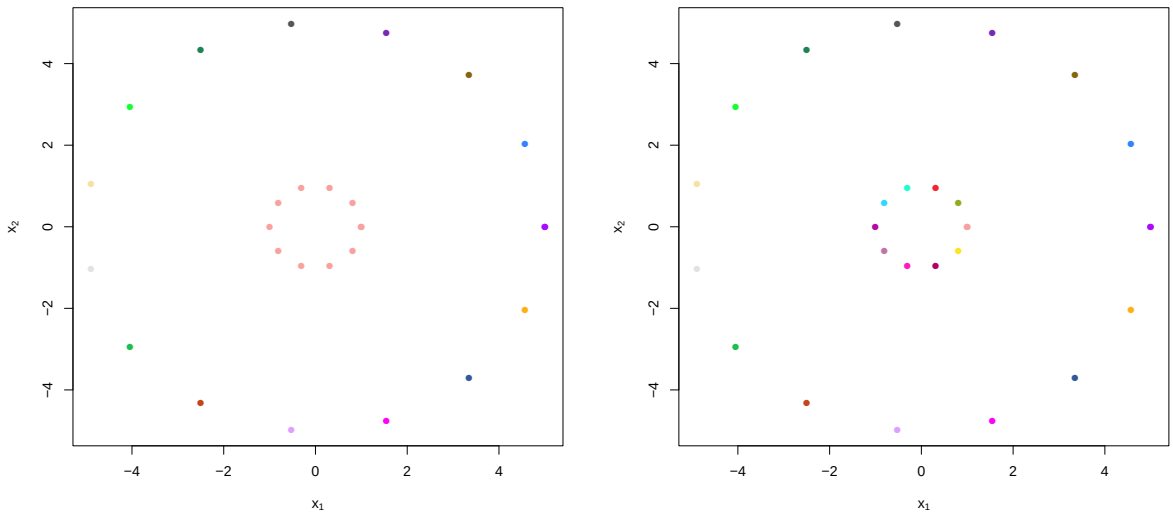
Наборы кластеров для различных значений $\alpha$.

```
colors <- palette('Alphabet')
clusters <- function(alpha)
{
  R <- Rz >= alpha

  CL <- R[1,]
```

```
  dim(CL) <- c(1,N)
  for (i in 2:N)
  {
    e <- 0
    for (j in 1:dim(CL)[1])
    {
      if (sum(CL[j,]!=R[i,])==0)
      {
        e <- 1
        break
      }
    }
    if (e == 0) CL <- rbind(CL,R[i,])
  }

  plot(X[,1],X[,2],type='n',
       xlab=expression(x[1]),ylab=expression(x[2]))
  M <- dim(CL)[1]
  for (i in 1:M)
    points(X[CL[i,],1],X[CL[i,],2],pch=16,col=colors[i])
}
```




```
clusters(0.8)    # порог 0.8 — один кластер в центре и кластеры-синглетоны снаружи
clusters(0.95)   # порог 0.95 — кластеры-синглетоны
```




## 6.4 Ассоциативные правила

### 6.4.1 Транзакции и наборы

Пусть имеется множество типов объектов $I = \{i_1, \ldots, i_N\}$ и множество наборов (транзакций) $T = \{T_1, \ldots, T_M\}$, где транзакция $T_i$ — некоторое подмножество элементов из $I$: $T_i = \{i_j^i \in I \mid j = 1, \ldots, n_i\}$, $i = 1, \ldots, M$.

Введём набор $F$ — некоторое подмножество элементов из $I$: $F = \{i_j^F \in I \mid j = 1, \ldots, n_F\}$. Обозначим через $T_F$ подмножество транзакций из $T$, содержащих $F$: $T_F = \{T_i \in T \mid F \subseteq T_i\}$. Поддержкой $\operatorname{supp}_F$ набора $F$ называется его относительная частота:

$$\operatorname{supp}_F = \frac{\operatorname{mes} T_F}{\operatorname{mes} T}.$$

Бывает полезно выделить наборы $F$, достаточно часто встречающиеся в $T$. Можно просто перебрать всевозможные подмножества $I$ (одноэлементные, двухэлементные, …, $N$-элементные) и отбросить те из них, поддержка которых меньше некоторого заданного порогового значения $\alpha$. Но при больших значениях $N$ количество таких подмножеств может быть очень большим:

$$\sum_{i=1}^{N} C_N^i.$$

Например, при $N = 10$ существует 5 120 комбинаций, а при $N = 20$ — уже 10 485 760. Уменьшить объём работы может наблюдение: если $F_i \subset F_j$, то $\operatorname{supp}_{F_i} \geqslant \operatorname{supp}_{F_j}$. На этом основан алгоритм Apriori:

1. $k \leftarrow 1$

   Построим набор $F^{(1)}$ одноэлементных множеств $F_j^{(1)} = \{i_j\}$, $j = 1, \ldots, N$.

2. Оставим в $F^{(k)}$ только те множества $F_j^{(k)}$, поддержка которых не меньше заданного порогового значения $\alpha$:

   $$F^{(k)} \leftarrow F^{(k)} \setminus \left\{F_j^{(k)} \in F^{(k)} \mid \operatorname{supp}_{F_j^{(k)}} < \alpha\right\}.$$

3. Если $\operatorname{mes} F^{(k)} \leqslant 1$, то завершаем выполнение алгоритма.

4. Объединяя те элементы $F_i^{(k)} \in F^{(k)}$ и $F_j^{(k)} \in F^{(k)}$, для которых $\mathrm{mes}(F_i^{(k)}, F_j^{(k)}) = k-1$, построим набор $(k+1)$-элементных множеств $F^{(k+1)}$.

5. $k \leftarrow k+1$

   Возвращаемся к шагу 2.

**Пример.** *Применение алгоритма Apriori.*

Генерирование исходных данных:

```
I <- 1:25          # типы объектов
N <- length(I)

M <- 1000          # число транзакций

T <- list()        # список транзакций
for (i in 1:M) T[[i]] <- sort(unique(trunc(runif(N,min(I),max(I+1)))))
```

Вспомогательные функции для вычисления частоты

```
freq <- function(F,T)
{
  s <- 0
  for (t in T) s <- s+ifelse(setequal(intersect(F,t),F),1,0)
  s
}
```

и поддержки

```
supp <- function(F,T) freq(F,T)/length(T)
```

Проверка, встречается ли $s$ в $F$:

```
exists <- function(s,F)
{
  for (f in F) if (setequal(s,f$collection)) return(TRUE)
  FALSE
}
```

Указываем пороговое значение и формируем часто встречающиеся наборы:

```
alpha <- 0.3       # пороговое значение

F <- list()        # список часто встречающихся наборов и их частот
F[[1]] <- list()   # список синглетонов и их частот
for (i in I) F[[1]][[i]] <- list(supp=supp(i,T),collection=i)

k <- 2
while (length(F[[k-1]])>1)    # Собственно Apriori
{
  L <- 1
  F[[k]] <- list()
  FF <- F[[k-1]]

  for (i in 1:(length(FF)-1))
    for (j in i:length(FF))
    {
      f <- sort(union(FF[[i]]$collection,FF[[j]]$collection))
      if (length(f)!=k || exists(f,F[[k]])) next
```

```
      s <- supp(f,T)
      if (s < alpha) next

      F[[k]][[L]] <- list(supp=s,collection=f)
      L <- L+1
    }

  k <- k+1
}
if (length(F[[length(F)]])==0) F <- F[-length(F)]
```

❑

### 6.4.2 Правила

Разделим $F$ на два непересекающихся множества: $A$ и $B$:

$$F = A \cup B, \quad A \cap B = \varnothing.$$

Будем называть *ассоциативным правилом* правило вида $A \to B$, означающее, что если $A \subset T_i$, то и $B \subset T_i$.

Характеристики правила $A \to B$.

- Поддержка (support)
$$\mathrm{supp}_{A\to B} = \frac{\mathrm{mes}\, T_{A\cup B}}{\mathrm{mes}\, T}.$$
- Достоверность (confidence)
$$\mathrm{conf}_{A\to B} = \frac{\mathrm{supp}_{A\to B}}{\mathrm{supp}_A}.$$
- Улучшение (improvement)
$$\mathrm{impr}_{A\to B} = \frac{\mathrm{supp}_{A\to B}}{\mathrm{supp}_A\, \mathrm{supp}_B}$$

Пусть у нас имеется некий часто встречающийся набор $F = \{i_1^F, \ldots, i_{n_F}^F\}$. Для построения всевозможных правил будем переберём всех чисел от 1 до $2^{n_F} - 2$, используя их представление в системе счисления с основанием 2. Если на $k$-м месте в бинарном представлении находится 1, то $i_k^F \in A$ , в противном случае — $i_k^F \in B$.

Преобразование $x$ в список бинарных значений:

```
num2bin <- function(x, len = 0)
{
  if (x==0) return(0)
  res <- c()
  while (x>0)
  {
     res <- c(res,x%%2)
     x <- x%/%2
  }
  if (len > length(res)) res <- c(res,rep(0,len-length(res)))
  res
}
```

Построение правил вида $A \to B$ с вычислением поддержки, достоверности и улучшения:

```
rules <- c()
N <- length(F)
for (i in 2:(2^N-2))
{
  bins <- num2bin(i,N)
  for (j in 1:length(F[[N]]))
  {
    A <- F[[N]][[j]]$collection[bins==1]
    B <- F[[N]][[j]]$collection[bins==0]
    rules[[length(rules)+1]] <-
       list(A=A,B=B,
            supp=F[[N]][[j]]$supp,
            conf=F[[N]][[j]]$supp/supp(A,T),
            impr=F[[N]][[j]]$supp/(supp(A,T)*supp(B,T)))
  }
}
```

Пример результата:

```
printrule <- function(rule)
{
  cat('rule␣(',rule$A,')␣->␣(',rule$B,')\n',
      '␣␣␣supp:␣',rule$supp,'␣␣␣conf:␣',rule$conf,
      '␣␣␣impr:␣',rule$impr,'\n')
}
for (rule in rules) printrule(rule)
```

```
rule ( 16 ) -> ( 6 20 )
    supp:  0.306    conf:  0.4622356    impr:  1.009248
rule ( 16 ) -> ( 13 20 )
    supp:  0.304    conf:  0.4592145    impr:  1.046047
.......
rule ( 16 20 ) -> ( 13 )
    supp:  0.304    conf:  0.6785714    impr:  1.029699
```

# Глава 7

# Имитационное моделирование

В настоящий момент, несмотря на более чем полувековое развитие системного анализа, нет одного общепринятого определения понятия «система». Проанализировав и объединив некоторые из них (см., например, [5: Гл. 1]), можно представить описание системы в виде следующего упорядоченного набора:

$$S = \langle A, R, G, T, E, N, L_N \rangle$$

- Структура
  - $A$ — составные части;
  - $R$ — связи.

  И составные части $A$, и связи $R$ обладают атрибутами $Q$, значения которых могут меняться в времени: $Q_A(t)$ и $Q_R(t)$.
- $G$ — цель. Имеется у всех искусственных систем (каждая из которых является средством для достижения этой цели), то есть систем, созданных людьми. У естественных систем имеется неявная цель (у живых существ цель — выживание) или же цели нет вообще (например, Солнечная *система* появилась только потому, что в определённом месте в определённый момент времени сложились подходящие условия для её возникновения).

  При наличии цели (явной или неявной) функционирование системы подчинено задаче достижения этой цели.

  Следует различать *декларируемые* цели и цели *действительные* (возможно, скрытые).
- Место и время
  - $E$ — окружение (среда). То, что влияет на систему (внешние воздействия $X$), и то, на что влияет система (отклик $Y$).
  - $T$ — временной интервал. Период времени или стадия развития, в течение которых изучается система.

    Например, человек может изучаться на протяжение всей его жизни ($T_1$). Или в его отрочестве $T_2 \subset T_1$ или в старости $T_3 \subset T_1$.

    В зависимости от выбора $T$ структура системы и её атрибуты могут рассматриваться как константы (короткие временные промежутки) или как функции от времени (продолжительные временны́е интервалы).

Функционирование системы можно представить в виде функции, связывающей значения на входе и на выходе в момент времени $t$:

$$S{:}\, Y = F(t, X), \qquad t \in T.$$

- $N$ — наблюдатель.

  Одну и ту же систему разные наблюдатели видят по разному. Один и тот же придорожный цветок для прохожий, учёный-ботаник, агроном и поэт/художник рассматривают с разных точек зрения.

  При описании наблюдателя следует указать его *цель* $G_N$: для чего производится наблюдение над данной системой.

  Для описания системы наблюдатель использует язык $L_N$. Иногда говорят, что нечто является системой тогда, когда для достаточно полного и адекватного её описания требуется использовать несколько языков.

Реальные системы $S$ (очень) часто имеет сложную структуру и поведение, затрудняющие работу с ними. *Модель* $S_M$ представляет собой упрощённое описание системы, в котором оставленно только самое важное с учётом $G_N$. Из внешних воздействий и откликов также выделяются только наиболее значимые и интересные с точки зрения наблюдателя подмножества $x \subset X$ и $y \subset Y$. Функция $f$ представляет собой упрощённое представление $F$:

$$S_M{:}\, f(t, x) = \tilde{y} \quad \approx y.$$

Здесь $\tilde{y}$ — отклик модели. Погрешностью модели назовём разность между реальным и модельным значениями:

$$e = y - \tilde{y}.$$

По способам представления модели и по её природе можно выделить

- Вербальные модели — описание системы на естественном языке.
- Графические модели — рисунки, графики, диаграммы и т.п.
- Физические модели — упрощённое физическое представление исследуемой системы.

  Вид физической модели может не совпадать с видом моделируемой системы (например, транспортные потоки в населённом пункте могут моделироваться гидродинамическими моделями из набора трубок различного диаметра).

- Абстрактные модели.
  - Математические модели — описание поведения системы набором математических уравнений, неравенств, логических условий и т.п.
  - Имитацинные модели — представление исследуемой системы в виде компьютерной программы.

## 7.1 Модели систем с непрерывным временем

В моделях с непрерывным временем состояние системы $S$ непрерывно зависит от времени $t \in T \subset \mathbb{R}$ [8: стр. 21]. В качестве примера можно привести задачу Коши ($f{:}\, \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$):

$$x'(t) = f\big(t, x(t)\big), \quad x(t_0) = x_0.$$

Если на временно́м интервале $T = [t_0, t_n]$ можно ввести такую сетку

$$t_0 < t_1 < \cdots < t_n,$$

что

$$x'(t) = f_i\big(t, x(t)\big), \qquad t_{i-1} \leqslant t < t_i, \quad i = 1, \ldots, n,$$

то такая модель называется гибридной. Также гибридной можно называть модель, в которой

$$x'(t) = f_i\big(t, x(t)\big) \text{ при } (x, x') \in X_i \subset \mathbb{R}^n \times \mathbb{R}^n.$$

Здесь $X_i$ — некоторое компактное множество — часть фазового пространства системы, на котором её поведение описывается дифференциальным уравнением с соответствующей функцией $f_i$.

**Пример** (Модель Лотка-Вольтерра «Хищник-Жертва»)**.** *Простейшая модель динамики численности популяций хищников (лисы) и жертв (кролики):*

$$r'(t) = \alpha r(t) - \beta r(t) f(t), \qquad r(t_0) = r_0 \tag{7.1a}$$
$$f'(t) = -\gamma f(t) + \delta r(t) f(t), \qquad f(t_0) = f_0 \tag{7.1b}$$

*Здесь*

- $r(t)$ — *объём популяции кроликов в момент времени* $t$*;*
- $f(t)$ — *объём популяции лис;*
- $\alpha$, $\beta$, $\gamma$ *и* $\delta$ — *положительные коэффициенты пропорциональности.*

```
pp <- function(t,x)
{
  alpha <- 1
  beta <- 0.01
  gamma <- 1
  delta <- 0.01

  r <- x[1]
  f <- x[2]

  c(alpha*r-beta*r*f, -gamma*f+delta*r*f)
}
```

Для численного решения задачи Коши

$$y'(x) = f\big(t, y(t)\big), \quad y(t_0) = y_0,$$

будем использовать метод Рунге-Кутты четвёртого-пятого порядка точности:

$$\begin{aligned}
y_{i+1} &= \frac{1}{6}(K_1 + 2K_2 + 2K_3 + K_4) \\
K_1 &= f(t_i, y_i), \\
K_2 &= f\left(t_i + \frac{h}{2}, y_i + \frac{h}{2}K_1\right), \\
K_3 &= f\left(t_i + \frac{h}{2}, y_i + \frac{h}{2}K_2\right), \\
K_4 &= f(t_i + h, y_i + hK_2).
\end{aligned}$$

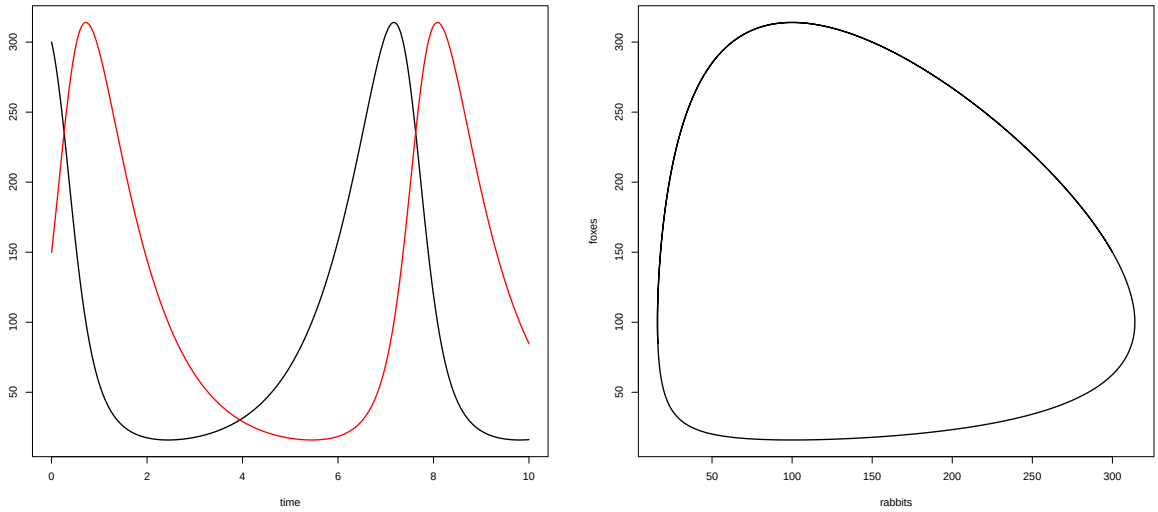
```
rk45 <- function(t,x,f,h)
{
  k1 <- f(t,x)
  k2 <- f(t+h/2,x+h/2*k1)
  k3 <- f(t+h/2,x+h/2*k2)
  k4 <- f(t+h,x+h*k3)
  1/6*(k1+2*k2+2*k3+k4)
}
```

Табличные значения функции на равномерной сетке с $n+1$ узлами с шагом $h$:

```
n <- 1000
h <- 0.01
t <- seq(from=0,by=h,length=n+1)
x <- array(dim=c(n+1,2))
x[1,] <- c(300,150)
for (i in 1:n) x[i+1,] <- x[i,]+h*rk45(t[i],x[i,],pp,h)
```

Графики динамики численности популяции от времени $r(t)$ и $f(t)$, а также график траектории в фазовом пространстве $(r, f)$:

```
plot(t,x[,1],ylim=c(min(x),max(x)),xlab='time',ylab='',type='n')
lines(t,x[,1],lwd=2)                # r(t)
lines(t,x[,2],lwd=2,col='red')      # f(t)

plot(x[,1],x[,2],xlab='rabbits',ylab='foxes',lwd=2,type='l')
```




Если в исходной модели (7.1a)–(7.1b) заменить в уравнении для кроликов (7.1a) часть с мальтузианской динамикой, предполагающей неограниченный рост,

$$r'(t) = \alpha r(t) + \dots$$

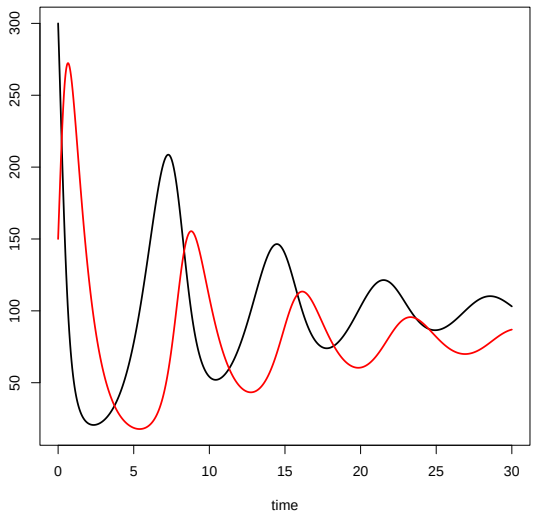
на логистическую динамику с максимальным значением $r_{\max}$:

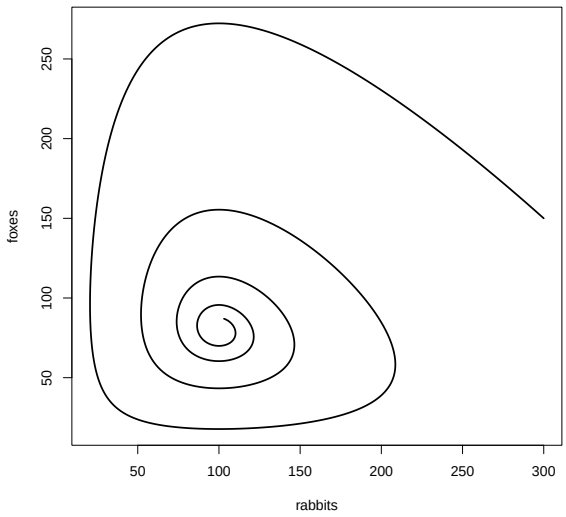
$$r'(t) = \alpha r(t)\left(1 - \frac{r(t)}{r_{\max}}\right) + \dots$$

то поведение решения качественно изменится.

```
pp <- function(t,x)
{
  alpha <- 1
```

```
  beta <- 0.01
  gamma <- 1
  delta <- 0.01

  r <- x[1]; r_max <- 500
  f <- x[2]

  c(alpha*r*(1-r/r_max)-beta*r*f, -gamma*f+delta*r*f)
}
```




## 7.2 Дискретно-временные модели

В дискретно-временных моделях временно́е множество дискретно: $T = \{t_0, t_1, \ldots, t_n\}$, где $t_{i-1} < t_i$, $i = 1, \ldots, n$. При изменении системного времени $t$ из $t_{i-1}$ в $t_i$ происходит изменение состояния системы:

$$S_{i-1} \to S_i.$$

**Пример** (Модель спроса и предложения)**.** *Предположим, что зависимости объёма спроса и предложения от цены описываются функциями* $Q_{спрос}(p)$ *и* $Q_{предложение}(p)$ *соответственно.*

*В момент времени* $t$ *для данной цены* $p_t$ *возможны три ситуации:*

- $Q_{спрос}(p_t) = Q_{предложение}(p_t)$ *и система находится в состоянии равновесия.*
- *Если* $Q_{спрос}(p_t) > Q_{предложение}(p_t)$*, то в системе имеется дефицит товара, что ведёт к увеличению цены на следующей итерации* $(t+1)$*.*
- *Отношение* $Q_{спрос}(p_t) < Q_{предложение}(p_t)$ *означает перепроизводство (избыток) товара, что также требует на следующей итерации* $(t+1)$ *коррекцию цены, только уже в сторону уменьшения.*

*Требуется построить модель динамики объёмов спроса и предложения, а также динамики цены, при условии, что*

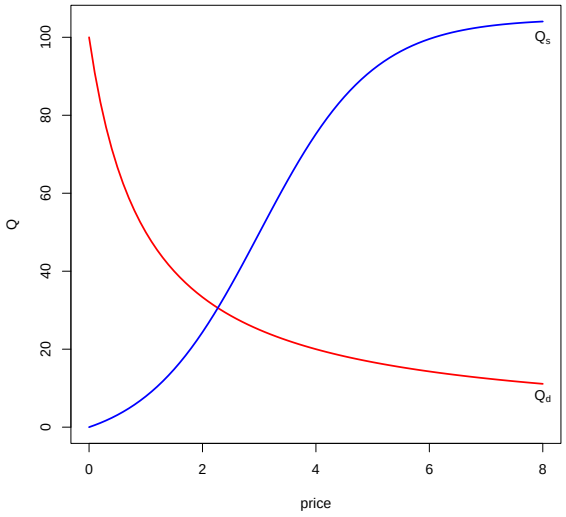
$$Q_{спрос}(p_{t-1}) = Q_{предложение}(p_t).$$

Пусть

$$Q_{\text{спрос}}(p) = \frac{Q_{\text{спрос,max}}}{p+1} \quad \text{и} \quad Q_{\text{предложение}}(p) = \frac{Q_{\text{предложение,max}}}{1+\exp(-p+\theta_1)} - \theta_2,$$

причём $Q_{\text{спрос,max}} > 0$, $Q_{\text{предложение,max}} > 0$ и $Q_{\text{предложение}}(0) = 0$. Например,

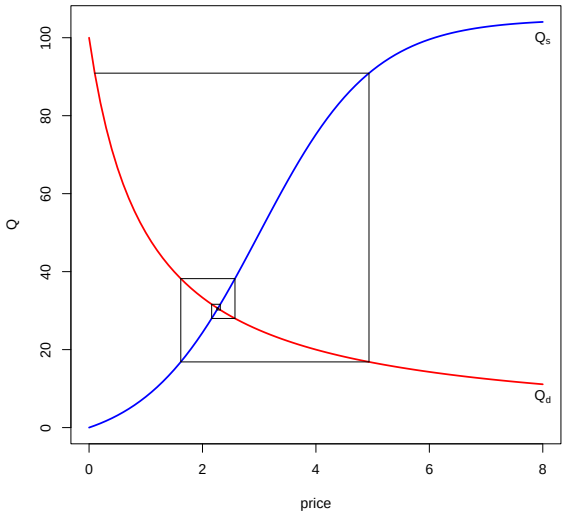
```
Qd <- function(p,q_max=100) q_max/(p+1)
Qs <- function(p,q_max=110) q_max/(1+exp(-p+3))-q_max/(1+exp(3))

P <- seq(from=0,to=8,by=0.1)
DS <- function() # графики зависимости спроса и предложения от цены
{
  plot(c(min(P),max(P)),c(min(Qs(P),Qd(P)),max(c(Qs(P),Qd(P)))),
       type='n',xlab='price',ylab='Q')
  lines(P,Qd(P),lwd=2,col='red')
  lines(P,Qs(P),lwd=2,col='blue')
  text(8,100,expression(Q[s]))
  text(8,8,expression(Q[d]))
}
DS()
```




Пусть при $t = 0$ цена товара равна $p_0$. Тогда

$$p_{t+1} = Q^{-1}_{\text{предложение}}\big(Q_{\text{спрос}}(p_t)\big).$$

```
eq <- function(x,p) Qs(x)-Qd(p)

n <- 10
price <- rep(0,n)
price[1] <- 1

library(nleqslv)
for (i in 2:n) price[i] <- nleqslv(price[i-1],eq,NULL,price[i-1])$x

# паутинообразный график
DS()
for(i in 2:n) lines(c(p[i-1],p[i],p[i]),
                    c(Qd(p[i-1]),Qs(p[i]),Qd(p[i])))
```

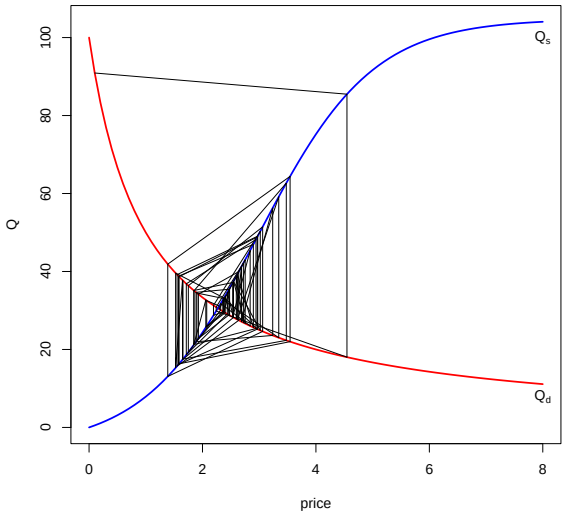



Предположим, что на значение цены в момент времени $t$ влияет случайный шум, и что значение цены не может быть меньше 0.1:

$$p_{t+1} = \max\Big(0.1, Q^{-1}_{\text{предложение}}\big(Q_{\text{спрос}}(p_t)\big) + \xi\Big), \quad \xi \sim \mathcal{N}(0, 0.5^2).$$

Тогда

```
n <- 50
p <- rep(0,n)
p[1] <- 0.1

for (i in 2:n)
{
  p[i] <- nleqslv(p[i-1],eq,NULL,p[i-1])$x
  p[i] <- p[i]+rnorm(1,sd=0.5)
}

DS()
for(i in 2:n) lines(c(p[i-1],p[i],p[i]),
                    c(Qd(p[i-1]),Qs(p[i]),Qd(p[i])))
```



❑

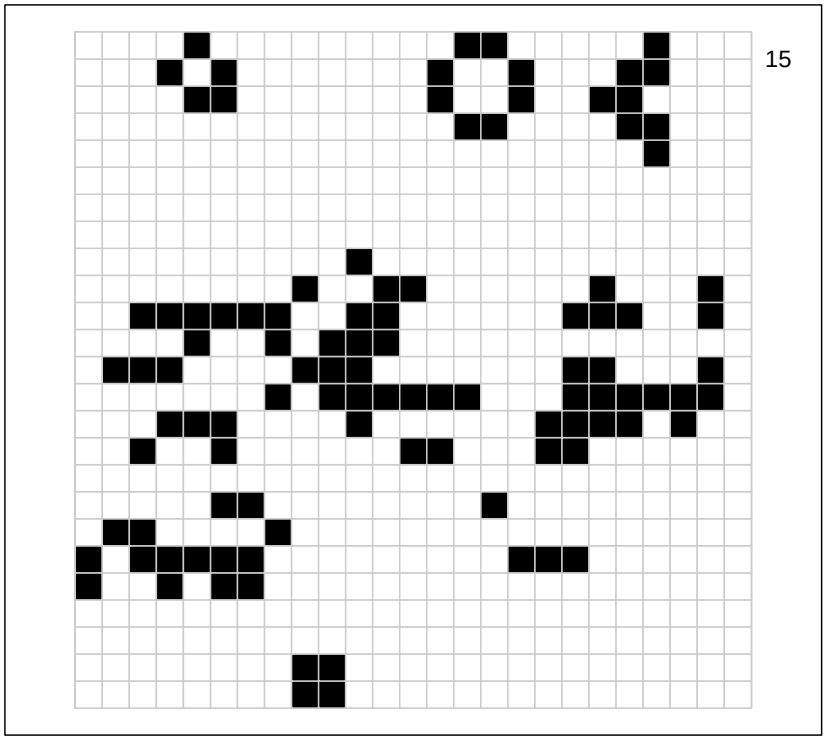
Вариант клеточного автомата, придуманный Джоном Конвеем.

**Пример** (Игра «Жизнь»). *В классическом варианте игры место действия — поле (конечное или бесконечное), поделённое на квадратные клетки. Каждая клетка может быть «живой» (значение 1) или «мёртвой» (значение 0); живые клетки на $i$-й итерации образуют $i$-е поколение популяции. Каждая клетка (за исключением расположенных на краю поля) имеет восемь соседей (соседних клеток), при этом (константы эволюции* B2/S23*)*

- *«мёртвая» клетка с трёмя живыми соседями на следующей итерации становится «живой»;*
- *если у «живой» клетки имеются два или три «живых» соседа, то на следующей итерации её состояние не меняется, в противном случае она «умирает» (меньше двух «живых» соседей — от одиночества, больше трёх — от перенаселения).*

```
m <- 25           # строк
n <- 25           # столбцов

born <- 3         # для возникновения жизни
live <- 2:3       # для сохранения жизни

# текущие значения
a <- array(rbinom(n*m,size=1,prob=.5),dim=c(m,n))
# следующие значения
a_next <- array(0,dim=c(m,n))

# количество «живых» соседей
neighborhood <- function(i,j)
  sum(a[max(1,i-1):min(m,i+1),max(1,j-1):min(n,j+1)])-a[i,j]

# здесь будем рисовать
plot(c(0,n),c(0,m),type='n',asp=1,
     xaxt='n',yaxt='n',ann=FALSE)

counter <- 1      # счётчик поколений
repeat
{
  for (i in 1:dim(a)[1])
    for (j in 1:dim(a)[2])
      rect((i-1),(j-1),i,j,
           col=ifelse(a[i,j]==0,'white','black'),
           border=rgb(.8,.8,.8))

  # пишем номер поколения
  rect(n+.5,m-1.5,n+2,m,col='white',border='white')
  text(n+1,m-1,counter)
  counter <- counter+1

  # следующее поколение
  for (i in 1:m)
    for (j in 1:n)
    {
      neighbors <- neighborhood(i,j)
```

```
      if (neighbors %in% born)
        a_next[i,j] <- 1
      else
        a_next[i,j] <- ifelse(neighbors %in% live,a[i,j],0)
    }

  # если конфигурация не изменилать, то заканчиваем моделирование
  if (sum(a!=a_next) == 0) break
  a <- a_next

  Sys.sleep(0.5) # задержка на полсекунды
}
```




Изменением функции, вычисляющей количество «живых» соседей, переходим с прямоугольного поля на цилиндр или тор (бесконечное, хотя и ограниченное поле):

```
# количество «живых» соседей: цилиндр
neighborhood <- function(i,j)
{
  x <- (i-1):(i+1); x[x<1] <- n; x[x>m] <- 1
  y <- max(1,j-1):min(n,j+1)
  sum(a[x,y])-a[i,j]
}

# количество «живых» соседей: тор
neighborhood <- function(i,j)
{
  x <- (i-1):(i+1); x[x<1] <- n; x[x>m] <- 1
  y <- (j-1):(j+1); y[y<1] <- n; y[y>m] <- 1
```

```
  sum(a[x,y])-a[i,j]
}
```

❑

Другой пример клеточного автомата.

**Пример** (Модель Т. Шеллинга сегрегации в городе). *В модели города Томаса Шеллинга город представляется как прямоугольный массив ячеек, каждая из которых обозначает занятый агентом или пустой дом.*

*Начальное распределение по домам случайно:*

- $p_i$ — *вероятность того, что дом занят* $i$*-м агентом* ($i = 1, \ldots, N$).
- $p_e$ — *вероятность того, что дом пуст.*

$$p_e + \sum_{i-1}^{N} p_i = 1, \qquad p_e, p_1, \ldots, p_N > 0.$$

*Например, если* $p_1 = p_2 = 0.45$, *то, очевидно,* $p_e = 0.1$.

*В каждый момент времени агент может быть счастлив или несчастлив в зависимости от своего окружения.*

- *Агент счастлив, если в его окружении имеется как минимум два (три, четыре) дома с похожими на агента соседями (или, возможно, пустых).*

  *В противном случае он несчастлив.*

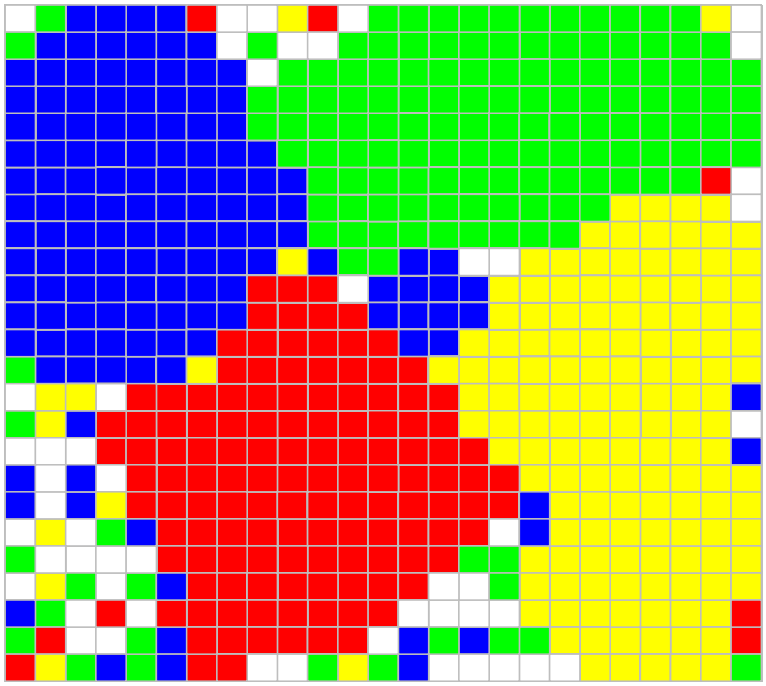
- *Если агент несчастлив, то он выбирает одну из незанятых ячеек и с вероятностью* $0 < p \leqslant 1$ *переезжает (можно считать, что* $p = 1$).

```
rm(list=ls(all=TRUE))

# Размер города:
nr <- 25   # nr — число строк
nc <- 25   # nc — число столбцов

threshold <- 3 # Пороговое значение для счастья

use_empty <- FALSE # Учитывать ли пустые дома при определении настроения агента

# Рисовать границы домов серым цветом
borderstyle <- 'grey'
# Не рисовать границы домов
#borderstyle <- FALSE

N <- 4  # Количество типов агентов
prob <- rep(0.9/N,N) # Одинаковые вероятности для всех агентов
                     # Вероятность пустого дома — 0.1

# Цвета для домов агентов разных типов
# Первый цвет для пустого дома
color <- c(rgb(1,1,1),
           rgb(1,0,0),rgb(0,1,0),rgb(0,0,1),
           rgb(1,1,0),rgb(1,0,1),rgb(0,1,1),
           rgb(.5,0,0),rgb(0,.5,0),rgb(0,0,.5),
```

```
            rgb(.5,.5,0),rgb(.5,0,.5),rgb(0,.5,.5))

# Исходные вероятности (ℝ)
T <- matrix(runif(nr*nc),nrow=nr,ncol=nc)
# Накопленные вероятности переводятся в номера типов агентов (ℕ)
for (i in 1:length(prob)) T[T<=sum(prob[1:i])] <- i+1
# Пустые дома
T[T<1] <- 1

# Координаты пустых домов
empty <- c()
for (i in 1:nr)
for (j in 1:nc)
  if (T[i,j]==1) empty <- c(empty,i,j)

# График исходного расположения агентов
plot(c(0,nc),c(0,nr),type='n',xlab='',ylab='',axes=FALSE)
for (i in 1:nc)
for (j in 1:nr)
  rect(i-1,j-1,i,j,
       col=color[T[i,j]],border=borderstyle)

# Определение настроения агента в (i,j)-м доме
good_mood <- function(i,j)
{
  # Определение количества соседей того же типа
  tmp <- T[max(1,i-1):min(nc,i+1),max(1,j-1):min(nr,j+1)]
  if (use_empty)
    s <- sum(tmp==T[i,j] | tmp==1)-1
  else
    s <- sum(tmp==T[i,j])-1

  # Определение настроения с учётом расположения дома на границе города
  if ((i==1 || i==nc) && (j==1 || j==nr))   # угловой дом
    s > min(threshold,3)
  else if (i==1 || i==nc || j==1 || j==nr) # дом на границе
    s > min(threshold,5)
  else
    s > threshold
}

# Генерация выборки размером n целочисленной случайной величины,
# значения которой равномерно распределены на множестве {1,2,...,r}
rdunif <- function(n,r) round(runif(n)*r+0.5)

counter <- 0 # Счётчик итераций
while (TRUE)
{
  counter <- counter+1

  # Агенты с плохим настроением
  bad_mood <- c()
  for (i in 1:nc)
```

```
  for (j in 1:nr)
  {
    if (T[i,j]==1 || good_mood(i,j)) next
    bad_mood <- c(bad_mood,i,j)
  }
  if (length(bad_mood)==0)
  {
    print('Все␣довольны')
    break
  }

  # Перераспределение агентов с плохим настроением
  # и обновление графика
  for (k in sample(seq(from=1,to=length(bad_mood),by=2)))
  {
    n <- rdunif(1,length(empty)/2)*2
    i <- bad_mood[k]
    j <- bad_mood[k+1]

    T[empty[n-1],empty[n]] <- T[i,j]
    rect(empty[n-1]-1,empty[n]-1,empty[n-1],empty[n],
         col=color[T[i,j]],border=borderstyle)

    T[i,j] <- 1
    if (length(empty)>2)
       empty <- empty[(1-n):(-n)]
    else
       empty <- c()
    empty <- c(empty,i,j)
    rect(i-1,j-1,i,j,col=color[1],border=borderstyle)
  }

  Sys.sleep(0.03)
}
cat(c('Выполнено␣итераций:␣',counter,'\n'))
```

Пример распределения агентов четырёх типов по городу $(25 \times 25)$ после $\approx 70$ итераций:



❑

## 7.3 Дискретно-событийные модели

В дискретно-событийных моделях временно́е множество $T$ также дискретно, но временна́я сетка $\{t_0, t_1, \ldots, t_n\}$ определяется тем, когда происходят события $e_i$, $i = 1, \ldots, n$.

**Пример** (Оптимизация затрат.)**.** *Производственная система состоит из $m$ станков, с каждым из которых изредка случаются поломки. Перед поломкой станок работает в течение промежутка времени, который является экспоненциально распределённой случайной величиной со средним значением 8 ч. Ремонтом станков занимаются $s$ ($s$ — неизменное положительное целое число) ремонтников. Для того чтобы починить один станок, одному ремонтнику требуется экспоненциально распределённый промежуток времени со средним значением 2 ч; с одним станком может работать только один ремонтник, даже если в это время есть ещё незанятые рабочие. Если в определённое время сломалось более $s$ станков, они образуют очередь на ремонт с дисциплиной FIFO и ожидают прибытия первого освободившегося ремонтника. Ремонтник полностью осуществляет ремонт станка, независимо от того, что ещё происходит в системе. Предположим, что каждый час простоя одного станка из-за поломки стоит системе 50 долларов, а час работы ремонтника — 10 долларов. (Ремонтники получают почасовую оплату, независимо от того, работали ли они в действительности.)*

*Пусть $m = 5$, однако предпочтительнее написать общий код, который путём изменения входного параметра способен обрабатывать значения вплоть до $m = 20$.*

*Требуется провести моделирование системы ровно при 800 ч. работы для каждой стратегии занятости $s = 1, 2, \ldots, m$, чтобы определить, какая из них даёт наименьшие ожидаемые средние расходы в час, допустив, что в начальный момент времени все машины только что прошли ремонт.*

Задача взята в [9: стр. 133, задача 1.22].

```
# m — количество станков
# s — число ремонтиков
# maxT — продолжительность моделирования
factory <- function(m,s,maxT)
{
  # Состояние станка
  stateWork <- 0     # работает
  stateInQueue <- 1  # в очереди на ремонт
  stateRepair <- 2   # ремонтируется

  workT <- 8    # Среднее время работы
  repairT <- 2  # Среднее время ремонта
  ZW <- 10      # Зарплата одному ремонтнику (в час)
  ZR <- 50      # Убытки из-за простоя станка (в час)

  # Начальная инициализация
  fr <- s                          # Все ремонтники свободны
  Z <- ZW * s * maxT               # Сразу начисляем зарплату всем работникам
  time_f <- 0                       # Начальный момент времени

  st <- rep(stateWork,m)           # Все станки работают
  tm <- rexp(m,rate=1/workT)       # Время работы до поломки
  queue <- c()                     # Очередь пуста

  while (TRUE)
  {
    # Ищем следующее событие
    machine <- which(tm==min(tm))
    time_f <- tm[machine]     # Сдвигаем системное время
    if (time_f > maxT) break

    # Изменяем состояние станка: работа→(очередь)→ремонт→работа
    if (st[machine] == stateWork) # работа→(очередь)→ремонт
    {
      # Все ремонтики заняты: помещаем станок в очередь на ремонт
      if (fr == 0)
      {
        st[machine] <- stateInQueue
        tm[machine] <- maxT + 1
        Z <- Z + (maxT - time_f) * ZR
        queue <- c(queue,machine)
      }
      # Есть свободные ремонтники: начинаем ремонт
      else
      {
        st[machine] <- stateRepair
        tm[machine] <- time_f + rexp(1,rate=1/repairT)
```

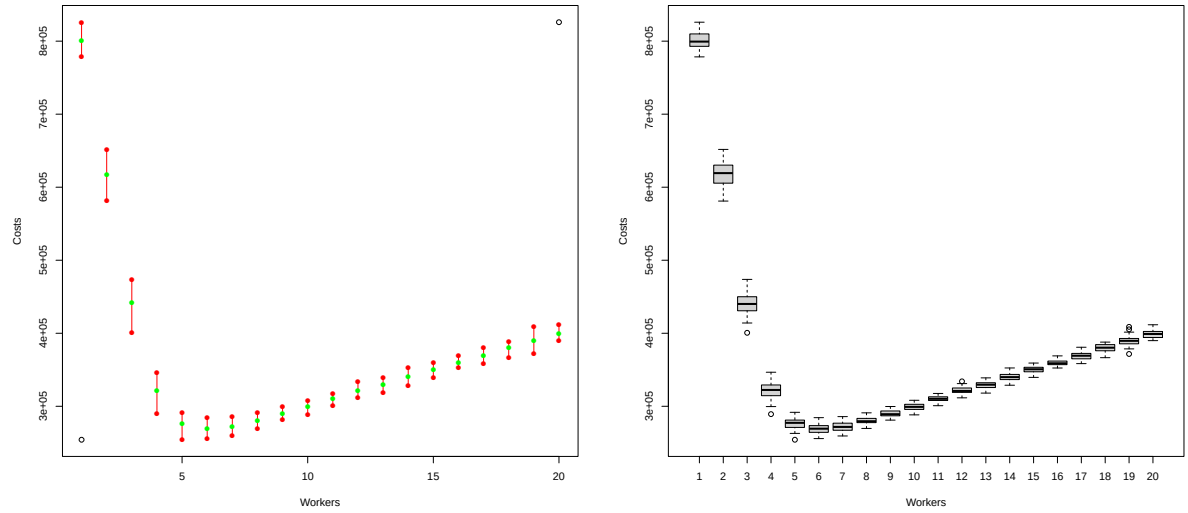
```
        if (tm[machine] > maxT)
          Z <- Z + (maxT - time_f) * ZR
        else
          Z <- Z + (tm[machine] - time_f) * ZR
        fr <- fr - 1
      }
    }
    else if (st[machine] == stateRepair) # ремонт→работа
    {
      st[machine] <- stateWork
      tm[machine] <- time_f + rexp(1,rate=1/workT)
      if (length(queue) == 0)                # Очередь пуста
        fr = fr + 1
      else
      {
        # Начинаем ремонтировать первый станок из очереди
        machine <- queue[1]
        st[machine] <- stateRepair
        tm[machine] <- time_f + rexp(1,rate=1/repairT)
        if (tm[machine] < maxT) Z <- Z - (maxT - tm[machine]) * ZR
        # Сдвигаем очередь
        if (length(queue) == 1)
          queue <- c()
        else
          queue <- queue[2:length(queue)]
      }
    }
  }

  Z # Возвращаем величину суммарных затрат
}
```

Для всех $s$ от 1 до 20 вычисляем среднее значение затрат по выборке из 50 значений.

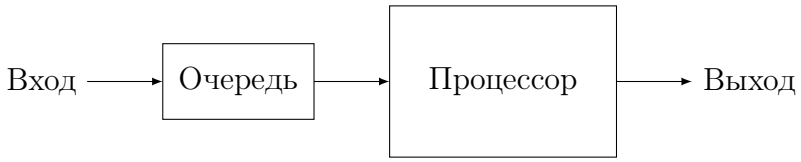
```
M <- 20 # станков
N <- 50 # объём выборки

Z <- rep(0,N*M) # массив результатов
dim(Z) <- c(N,M)

for (s in 1:M)
{
  for (i in 1:N) Z[i,s] <- factory(M,s,1000)
}

# минимальные, средние и максимальные значения для каждого s
stat <- rep(0,M*3); dim(stat)<-c(3,M)
for (s in 1:M)
{
  stat[1,s] <- min(Z[,s])
  stat[2,s] <- mean(Z[,s])
  stat[3,s] <- max(Z[,s])
}

plot(c(1,M),c(min(stat),max(stat)),xlab="Workers",ylab="Costs")
```

```
for (s in 1:M) lines(c(s,s),stat[c(1,3),s],col='red',type='o',pch=16)
points(1:M,stat[2,],col="green",pch=16)

# квантили
Zpl <- NULL
Zfact <- NULL
for (s in 1:M) for (i in 1:N)
{
  Zpl <- c(Zpl,Z[i,s])
  Zfact <- c(Zfact,s)
}
Zfact <- as.factor(Zfact)
plot(Zfact,Zpl,xlab='Workers',ylab='Costs')
```

Результат:




**Пример** (Модель процессора)**.**



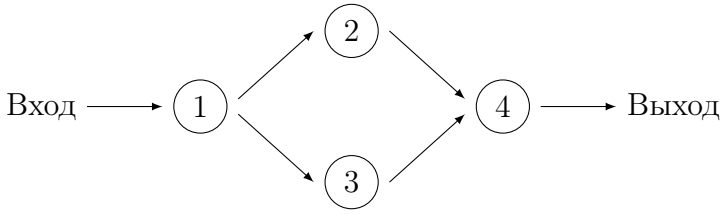

```
# процессор

tau_p <- 4    # ожидаемое время обработки транзакта
t_p   <- -1   # время окончания текущего действия
qlen  <- 0    # длина очереди

# модель

T <- 100      # продолжительность моделирования
t <- 0        # текущее модельное время

# транзакты

tau_t <- 5    # ожидаемое время между транзактами
```

```
t_t <- t + rexp(1, 1/tau_t) # время появления следующего транзакта

# история событий
events <- list(t = t, qlen = qlen, state = ifelse(t_p==-1,0,1))

# моделирование
while (t <= T)
{
  if (t_p == -1 || t_p > t_t)
  {
    # появляется новый транзакт

    t <- t_t
    t_t <- t + rexp(1, 1/tau_t)

    if (t_p == -1)
      t_p <- t + rexp(1, 1/tau_p)
    else
      qlen <- qlen + 1
  }
  else
  {
    # заканчивается обработка транзакта

    t <- t_p

    if (qlen == 0)
      t_p <- -1
    else
    {
      t_p <- t + rexp(1, 1/tau_p)
      qlen <- qlen - 1
    }
  }

  # добавляем новое событие в историю
  events$t     <- c(events$t, t)
  events$qlen  <- c(events$qlen, qlen)
  events$state <- c(events$state, ifelse(t_p==-1,0,1))
}

# корректируем последнее событие

n <- length(events$t)
events$t[n]     <- T
events$qlen[n]  <- events$qlen[n-1]
events$state[n] <- events$state[n-1]

# график длины очереди в зависимости от времени
plot(events$t, events$qlen, type='s', lwd=3, col='blue',
               xlab='t',ylab='query')
# график состояния процессора в зависимости от времени
lines(events$t, events$state, type='s', lwd=1, col='red')
```

```
# максимальная длина очереди
max(events$qlen)
# средняя длина очереди
sum(events$qlen[1:(n-1)]*(events$t[-1]-events$t[-n])) / T

# доля времени, когда процессор был занят
sum(events$state[1:(n-1)]*(events$t[-1]-events$t[-n])) / T
```

❑

**Пример** (Модель сети процессоров).




```
# модель

T <- 100
t <- 0

# транзакты

next_req_time <- function(t) t + rexp(1, 1/5)
req_time <- next_req_time(t)

# процессор

as.processor <- function(id, est_time=1, time=-1,
                         qlen=0, descendants=c())
{
  p <- list(id = id,                  # идентификатор
            get_time=function(t) t + rexp(1,1/est_time),
            time=time,                # время завершения обработки
            qlen=qlen,                # длина очереди
            descendants=descendants,# следующие процессоры
            events=list(t=t,          # история
                        qlen=qlen,
                        state=ifelse(time==-1,0,1)))

  class(p) <- 'processor'
  p
}

is.processor <- function(x) inherits(x, 'processor')

print.processor <- function(x)
{
  cat(paste0('Processor #',x$id,'\n'),
      '  Time =',x$time,'  Query =',x$qlen,'\n')
```

```
}

# сеть процессоров

procs <- list()

procs[[1]] <- as.processor(1, est=4, desc=c(2,3))
procs[[2]] <- as.processor(2, est=7, desc=c(4))
procs[[3]] <- as.processor(3, est=7, desc=c(4))
procs[[4]] <- as.processor(4, est=4)

# процессор proc[[ind]] получает транзакт
req_in <- function(ind, t)
{
  if (procs[[ind]]$time == -1)
  {
    procs[[ind]]$time <<- procs[[ind]]$get_time(t)
  }
  else
  {
    procs[[ind]]$qlen <<- procs[[ind]]$qlen + 1
  }

  proc_update(ind,t)
}
# процессор proc[[ind]] завершает обработку транзакта
req_out <- function(ind, t)
{
  if (procs[[ind]]$qlen == 0)
  {
    procs[[ind]]$time <<- -1
  }
  else
  {
    procs[[ind]]$time  <<- procs[[ind]]$get_time(t)
    procs[[ind]]$qlen <<- procs[[ind]]$qlen - 1
  }

  if (length(procs[[ind]]$desc)>0)
  {
     query <- sapply(procs[procs[[ind]]$desc],function(p) p$qlen)
     nxt <- which(query==min(query))[1]
     req_in(procs[[ind]]$desc[nxt], t)
     print(procs[[procs[[ind]]$desc[nxt]]])
  }

  proc_update(ind,t)
}
# процессор proc[[ind]] обновляет историю событий в момент времени t
proc_update <- function(ind, t)
{
  procs[[ind]]$events$t     <<- c(procs[[ind]]$events$t, t)
  procs[[ind]]$events$qlen  <<-
```

```
      c(procs[[ind]]$events$qlen, procs[[ind]]$qlen)
  procs[[ind]]$events$state <<-
      c(procs[[ind]]$events$state, ifelse(procs[[ind]]$time==-1,0,1))
}

# следующее событие (индекс процессора, заканчивающего обработку, или
# -1 при появлении нового транзакта)
next_event <- function()
{
  times <- sapply(procs[1:length(procs)], function(p) p$time)
  if (sum(times>0)>0)
  {
    ind <- which(times==min(times[times>0]))[1]
    ifelse(procs[[ind]]$time<req_time,ind,-1)
  }
  else -1
}

# моделирование
while (t <= T)
{
  ind <- next_event()

  if (ind == -1)
  {
    t <- req_time
    req_time <- next_req_time(t)
    req_in(1, t)
  }
  else
  {
    t <- procs[[ind]]$time
    req_out(ind, t)
  }

  print(procs[[ifelse(ind>0,ind,1)]])
}
```

❑

## 7.4 Стохастическая оптимизация

При работе с имитационными моделями часто возникает задача поиска таких параметров модели, при которых некоторый критерий (при описании системы со случайными параметрами имеющий стохастическую природу) достигал бы своего экстремального значения.

Пусть требуется решить задачу безусловной минимизации функции $f\colon \mathbb{R}^m \to \mathbb{R}$ со стохастическим параметром $\xi$

$$f(x,\xi) \to \min$$

или условной минимизации, когда в дополнение к целевой функции $f$ имеется содержащее стохастический параметр $\eta$ ограничение $g\colon \mathbb{R}^m \to \mathbb{R}^n$:

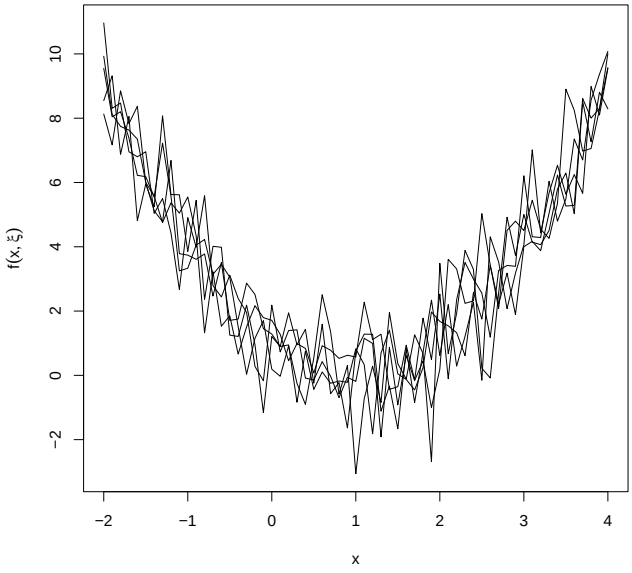
$$f(x,\xi) \to \min, \quad g(x,\eta) \leqslant 0.$$

В качестве элементарного примера возьмём функцию $f(x) = (x-1)^2 + \xi$, где $\xi \sim \mathcal{N}(0,1)$.

```
f <- function(x) (x-1)^2+rnorm(length(x),sd=1)
```

Ниже приведён график нескольких реализаций функции $f$ при $x \in [-2,4]$.

```
x <- seq(from=-2,to=4,by=0.1)
graphics <- function(n=5)
{
  y <- array(0,dim=c(n,length(x)))
  for (i in 1:n) y[i,] <- f(x)

  plot(c(min(x),max(x)),c(min(y),max(y)),
       xlab='x',ylab=expression(f(x,xi)),type='n')
  for (i in 1:n) lines(x,y[i,])
}
graphics()
```




Очевидно, что в исходной постановке задача не имеет решения, так как нет единственного общепринятого способа сравнения случайных величин $f(x,\xi)$ и $f(x',\xi)$.

Для сведéния стохастичесой задачи к детерминированной форме можно воспользоваться подходами, подробно описанными, например, в [11].

### 7.4.1 Модель ожидаемого значения

При сравнении случаных величин $\xi$ и $\eta$ предполагаем, что

$$\mathsf{E}\,\xi \leqslant \mathsf{E}\,\eta \;\vdash\; \xi \preceq \eta.$$

Таким образом стохастическая задача безусловной минимизации заменяется на

$$\mathsf{E}\,f(x,\xi) \to \min,$$

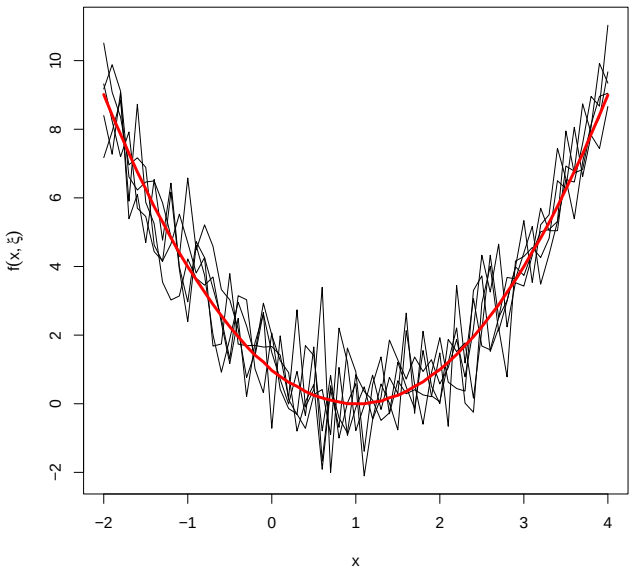
а условной минимизации — на

$$\mathsf{E}\,f(x,\xi) \to \min, \quad \mathsf{E}\,g(x,\eta) \leqslant 0.$$

```
Ef <- function(x,n) mean(f(rep(x,n)))   # E f(x,ξ) ≈ f̄(x)
```

График $\bar f$ (красная линия) на фоне пучка частных реализаций $f(x,\xi)$.

```
graphics()
n <- 10000
lines(x,sapply(x,Ef,n),lwd=3,col='red')
```




```
optimize(Ef,c(-2,4),n)   # x̃_opt = 1.065687,  f̄(x̃_opt) = 0.00657364
optimize(Ef,c(-2,4),n)   # x̃_opt = 1.002164,  f̄(x̃_opt) = 0.00434226
optimize(Ef,c(-2,4),n)   # x̃_opt = 0.9595278, f̄(x̃_opt) = 0.003414021
```

### 7.4.2 Модель с вероятностными ограничениями

При сравнении случаных величин $\xi$ и $\eta$ предполагаем, что при заданном значении вероятности $\alpha$

$$\mathsf{P}\{\xi \geqslant \xi_\alpha\} = \alpha, \quad \mathsf{P}\{\eta \geqslant \eta_\alpha\} = \alpha$$

$$\xi_\alpha \leqslant \eta_\alpha \;\vdash\; \xi \preceq \eta.$$

Исходная задача безусловной стохастической минимизации сводится к минимизации функции $f^\star(x) = f(x,\xi)_{\alpha_f}$

$$\mathsf{P}\big\{f(x,\xi) \leqslant f^\star(x)\big\} = \alpha_f, \quad f^\star(x) \to \min,$$

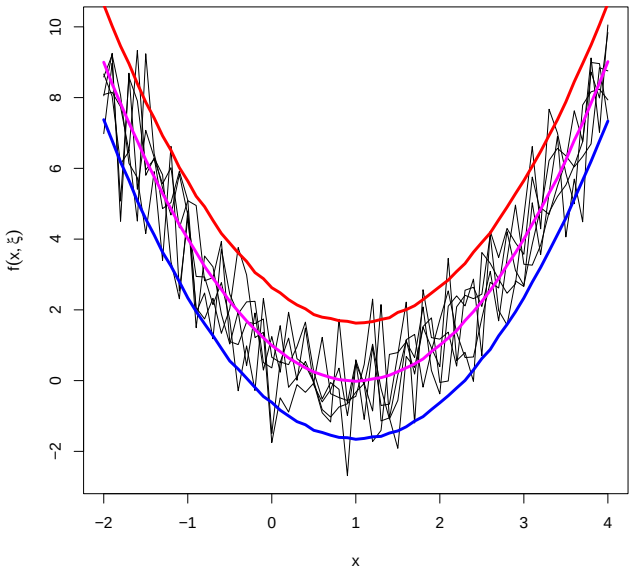
а условной минимизации — к задаче

$$\mathsf{P}\big\{f(x,\xi) \leqslant f^\star(x)\big\} = \alpha_f, \quad f^\star \to \min$$
$$\mathsf{P}\big\{g(x,\xi) \leqslant 0\big\} = \alpha_g.$$

```
Pf <- function(x,alpha,n) quantile(f(rep(x,n)),alpha)
```

Графики $f^\star(x)$ на фоне пучка частных реализаций $f(x,\xi)$ при $\alpha_f = 0.95$ (красный), $\alpha_f = 0.5$ (пурпурный) и $\alpha_f = 0.05$ (синий).

```
graphics()
n <- 10000
lines(x,sapply(x,Pf,0.95,n),lwd=3,col='red')      # ξ_0.95
lines(x,sapply(x,Pf,0.5,n),lwd=3,col='magenta')   # ξ_0.5 (медиана)
lines(x,sapply(x,Pf,0.05,n),lwd=3,col='blue')     # ξ_0.05
```




```
alpha <- 0.95
optimize(Pf,c(-2,4),alpha,n)  # x̃_opt = 1.034379, P{f(x_opt,ξ) ⩽ 1.653534} ⩾ 0.95
optimize(Pf,c(-2,4),alpha,n)  # x̃_opt = 1.011047, P{f(x_opt,ξ) ⩽ 1.642613} ⩾ 0.95
optimize(Pf,c(-2,4),alpha,n)  # x̃_opt = 0.8831557, P{f(x_opt,ξ) ⩽ 1.658979} ⩾ 0.95
```

### 7.4.3 Событийное моделирование

$$\mathsf{P}\{f(x,\xi) \leqslant f^\star\} \to \max$$

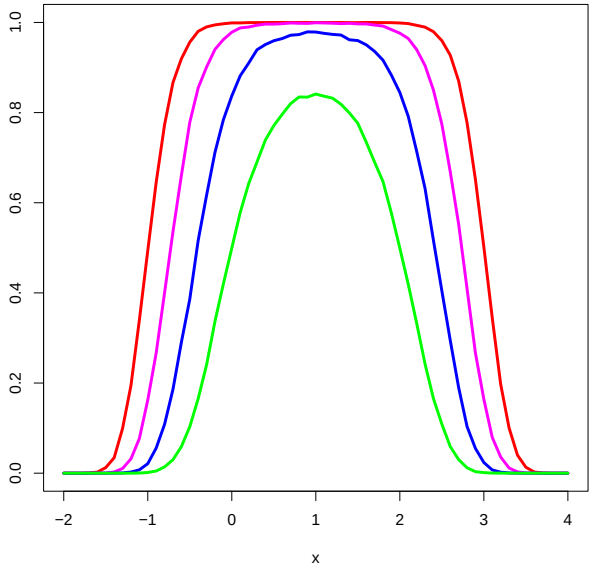
$$\mathsf{P}\{f(x,\xi) \leqslant f^\star\} \to \max$$

$$\mathsf{P}\{g(x,\xi) \leqslant 0\} \geqslant \alpha_g$$

```
Pf <- function(x,fs,n) sum(f(rep(x,n))<=fs)/n
```

Графики $\mathsf{P}\{f(x,\xi) \leqslant f^\star\}$ при различных значениях $f^\star$ (при $f^\star = 4$ график красного цвета, при $f^\star = 3$ — пурпурного, при $f^\star = 2$ — синего и при $f^\star = 1$ — зелёного).

```
n <- 10000
plot(c(min(x),max(x)),c(0,1),
     xlab='x',ylab='',type='n')
fs <- c(4,3,2,1)
color <- c('red','magenta','blue','green')
for (i in 1:length(fs))
  lines(x,sapply(x,Pf,fs[i],n),lwd=3,col=color[i])
```




```
optimize(Pf,c(-2,4),2,n,maximum=TRUE)
```
$\#\ \tilde{x}_{opt} = 0.9311438,\ \mathsf{P}\big\{f(x_{opt},\xi) \leqslant 2\big\} \approx 0.9768$

```
optimize(Pf,c(-2,4),2,n,maximum=TRUE)
```
$\#\ \tilde{x}_{opt} = 0.9954182,\ \mathsf{P}\big\{f(x_{opt},\xi) \leqslant 2\big\} \approx 0.9805$

```
optimize(Pf,c(-2,4),2,n,maximum=TRUE)
```
$\#\ \tilde{x}_{opt} = 0.9954182,\ \mathsf{P}\big\{f(x_{opt},\xi) \leqslant 2\big\} \approx 0.975$

# Список литературы


1. *Абельсон Х.*, *Сассман Д. Д.* Структура и интерпретация компьютерных программ: Пер. с англ. — Добросвет, КДУ, 2018. — 608 с. — (Цит. на с. 86, 100).
2. *Бокс Д.*, *Дженкинс Г. М.* Анализ временных рядов: прогноз и управление: Пер. с англ. Ч. 1. — М. : Мир, 1974. — 405 с. — (Цит. на с. 157).
3. *Боровков А. А.* Теория вероятностей: Учеб. пособие для вузов. — 2-е изд., перераб. и доп. — М. : Наука. Гл. ред. физ.-мат. лит., 1986. — 432 с. — (Цит. на с. 43).
4. *Васильев Ф. П.*, *Иваницкий А. Ю.* Линейное программирование. — М. : Факториал Пресс, 2003. — 352 с. — (Цит. на с. 66).
5. *Волкова В. Н.*, *Денисов А. А.* Теория систем: Учеб. пособие. — М. : Высшая школа, 2006. — 511 с. — (Цит. на с. 194).
6. *Голуб Д.*, *Ван Лоун Ч.* Матричные вычисления: Пер. с англ. — М. : Мир, 1999. — 548 с. — (Цит. на с. 31).
7. *Джонс М. Т.* Программирование искусственного интеллекта в приложениях: Пер. с англ. — М. : ДМК Пресс, 2004. — 312 с. — (Цит. на с. 72, 75).
8. *Карпов Ю.* Имитационное моделирование систем. Введение в моделирование с AnyLogic 5. — СПб. : БХВ-Петербург, 2005. — 400 с. — (Цит. на с. 195).
9. *Кельтон Д. В.*, *Лоу А. М.* Имитационное моделирование. Классика CS: Пер. с англ. — 3-е изд. — СПб. : Питер, 2004. — 847 с. — (Цит. на с. 207).
10. *Кобзарь А. И.* Прикладная математическая статистика. Для инженеров и научных работников. — М. : ФИЗМАТЛИТ, 2006. — 816 с. — (Цит. на с. 41, 42).
11. *Лю Б.* Теория и практика неопределённого программирования: Пер. с англ. — М. : БИНОМ. Лаборатория знаний, 2005. — 416 с. — (Цит. на с. 214).
12. *Магнус Я. Р.*, *Катышев П. К.*, *Пересецкий А. А.* Эконометрика. Начальный курс: Учеб. — 6-е изд., перераб. и доп. — М. : Дело, 2004. — 576 с. — (Цит. на с. 104, 130, 147, 150).
13. Методы и модели анализа данных: OLAP и Data Mining / А. А. Барсегян [и др.]. — СПб. : БХВ-Петербург, 2004. — 336 с. — (Цит. на с. 187).
14. Многомерный статистический анализ в экономике: Учеб. пособие для вузов / Л. А. Сошникова [и др.]. — М. : ЮНИТИ-ДАНА, 1999. — 598 с. — (Цит. на с. 55).
15. *Реклейтис Г.*, *Рейвиндран А.*, *Рэгсдел К.* Оптимизация в технике: В 2-х кн. Кн. 1: Пер. с англ. — М. : Мир, 1986. — 349 с. — (Цит. на с. 68—70, 72, 138).
16. Справочник по специальным функциям с формулами, графиками и математическими таблицами: Пер. с англ. / под ред. М. Абрамовица, И. Стиган. — М. : Наука, 1979. — 832 с. — (Цит. на с. 29).
17. *Траск Э.* Грокаем глубокое обучение: Пер. с англ. — СПб. : Питер, 2019. — 352 с. — (Цит. на с. 174).

18. *Фаддеев Д.*, *Фаддеева В.* Вычислительные методы линейной алгебры. — СПб. : Лань, 2002. — 736 с. — (Цит. на с. 33).

19. *Хайкин С.* Нейронные сети: полный курс: Пер. с англ. — Изд. 2-е. — М. : Изд. дом «Вильямс», 2006. — 1104 с. — (Цит. на с. 174, 179, 180).

20. *Хендерсон П.* Функциональное программирование. Применение и реализация: Пер. с англ. — М. : Мир, 1983. — 349 с. — (Цит. на с. 100).

21. *Хилл Ф.* OpenGL. Программирование компьютерной графики. Для профессионалов: Пер. с англ. — СПб. : Питер, 2002. — 1088 с. — (Цит. на с. 132).

22. *Хорн Р.*, *Джонсон Ч.* Матричный анализ: Пер. с англ. — М. : Мир, 1989. — 655 с. — (Цит. на с. 31).

23. *Шафаревич И. Р.* Основные понятия алгебры, Алгебра 1. Т. 11. — М. : ВИНИТИ, 1986. — С. 5—279. — (Итоги науки и техн. Сер. Соврем. пробл. мат. Фундам. направления). — (Цит. на с. 17).

24. *Шолле Ф.* Глубокое обучение на R: Пер. с англ. — СПб. : Питер, 2018. — 400 с. — (Цит. на с. 174).

25. *Cichosz P.* Data Mining Algorithms: Explained Using R. — John Wiley & Sons, 2015. — 683 p. — (Cit. on p. 173).

26. *Makhabel B.* Learning Data Mining with R. — Packt Publishing Ltd, 2015. — 287 p. — (Cit. on p. 173).

27. *Russel S. J.*, *Norvig P.* Artifical Intelligence. A Modern Approach. — Prentice Hall, 2010. — 1132 p. — (Cit. on p. 87).

28. *Venables W. N.*, *Smith D. M.*, *Team R. C.* An Introduction to R. Notes on R: A Programming Environment for Data Analysis and Graphics. — 2022. — 105 p. — (Cit. on p. 140).

29. *Wickham H.* ggplot2: Elegant Graphics for Data Analysis. — Springer-Verlag New York, 2016. — 212 p. — (Cit. on p. 129).

# Предметный указатель

**Шишкин Владимир Андреевич**

Кандидат физико-математических наук

Доцент кафедры информационных систем и математических методов в экономике Экономического факультета Пермского государственного национального исследовательского университета

Для предложений, замечаний, критики, указания на ошибки — e-mail: vsh1791@mail.ru

*Учебное издание*

**Шишкин** Владимир Андреевич

# Математические пакеты: R

Учебное пособие

Редактор *Н. И. Стрекаловская*
Корректор *И. Н. Цветкова*
Компьютерная верстка: *В. А. Шишкин*

---

Объем данных 15,0 Мб
Подписано к использованию 25.12.2023

---

Размещено в открытом доступе
на сайте www.psu.ru
в разделе НАУКА / Электронные публикации
и в электронной мультимедийной библиотеке ELiS

Управление издательской деятельности
Пермского государственного
национального исследовательского университета
614068, г. Пермь, ул. Букирева, 15