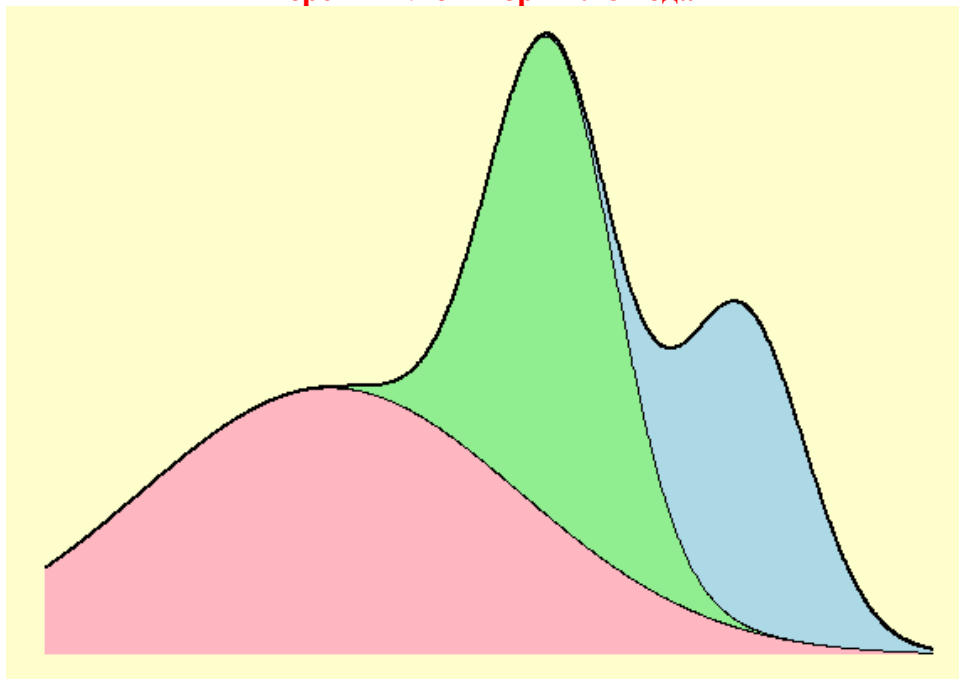


СПРАВОЧНИК ПО БАЗОВЫМ КОМАНДАМ СИСТЕМЫ R

Дьяконов Александр

**ВНИМАНИЕ! ДОКУМЕНТ НАХОДИТСЯ В СТАДИИ ПОСТОЯННОЙ ПРАВКИ!
ВОЗМОЖНЫ ОШИБКИ!**

Версия «19 октября 2013 года»



Рекомендуемые книги, электронные ресурсы:

- http://zoonek2.free.fr/UNIX/48_R/all.html Vincent Zoonekynd **Statistics with R** Один из самых полных источников по R.
- <http://nostarch.com/artofr.htm> Norman Matloff **The Art of R programming** Полноценный учебник по R.
- <http://r-analytics.blogspot.com/> Советы по установке, использованию, ссылки на книги
- <http://ru.wikipedia.org/wiki/> Много полезных ссылок.

СПИСОК БУДЕТ ПОПОЛНЕН.

Особенности системы

- [+] Бесплатное ПО, содержащее все необходимые функции для обработки, визуализации и анализа данных.
- [+] Превосходная реализация многих алгоритмов анализа данных. Например, одна из лучших реализаций алгоритма случайного леса (RF).
- [-] Требуется загрузки всех данных в память!
- [-] Является интерпретатором с не очень богатыми возможностями по визуализации данных.

Особенности этого учебного пособия

Автор считает, что лучше всего подобные системы изучать на примерах. Ведь знакомятся с ними люди, уже освоившие один или даже несколько высокоуровневых языков программирования. Синтаксис учить просто, а все тонкости всё равно познаются в процессе работы. Необходимо лишь показать примеры работы с системой. Это должны быть маленькие и самостоятельные фрагменты кода, из которых при желании можно составлять большие программы для решения достаточно сложных задач.

Многие примеры взяты из [1]–[2]. Большая часть из них существенно переработана и изменена. Автор позволил себе не указывать каждый раз источник, но очень благодарен всем, кто пишет дельные книги по системе R.

Что сразу надо учитывать

R сочетает объектно-ориентированный и функциональный подходы к программированию.

R – скриптовый язык, поэтому, в частности, в нём нет объявления переменных.

Установка

Скачать новую версию можно с сайта <http://cran.r-project.org>. При установке не должно возникнуть проблем.

Основная мощь системы заключена в дополнительных пакетах, которые устанавливаются отдельно (например, командой **install.packages**, см. ниже).

На компьютере могут отдельно сосуществовать несколько версий системы (для каждой нужно будет устанавливать пакеты).

Начало. Прimitives операции

<pre>> help(solve) > ?solve > help(" ")</pre>	<p>Вызов помощи.</p> <p>Попробуйте набрать example(persp). Многие функции имеют хорошие примеры своей работы!</p>
<pre>> help.start()</pre>	<p>Вызов html-помощи.</p>
<pre>> x <- 15 # можно x=15 > x Ошибка: объект "A" не найден > x[1] [1] 15</pre>	<p>Присваивание.</p> <p>Чувствительность к регистру!</p> <p>Скаляр – это вектор единичной длины. Элементы вектора нумеруются с единицы.</p>
<pre>> x<-y<-z<-0 > y [1] 0 > ПервПер <- 2 -> ВторПер > ВторПер [1] 2</pre>	<p>Сложные присваивания.</p> <p>Допустимы и такие присваивания. Допустима кириллица! В R вообще сделано максимум для того, чтобы код программы читался как обычный текст.</p>
<pre>> a <- c(1,3,4) > a = c(1,3,4) > c(1,3,4) -> a > assign("a", c(1,3,4)) > a [1] 1 3 4</pre>	<p>Различные способы присваивания.</p> <p>Обратите внимание на кавычки!</p>
<pre>> a = 1; b = 2; a+b #комментарий [1] 3 > c = a-b > (c = a-b) [1] -1 > (c = + a - b) [1] -1</pre>	<p>Оформление. Команды отделяются точкой с запятой или переводом строки. В конце команды точка с запятой не обязательна. Команда в скобках выводит результат.</p> <p>Если команда не завершена, выводится «приглашение» «+». Этот приём часто используется при написании листингов.</p>
<pre>> a = {x = 1; y = 1;</pre>	<p>Блок команд выделяется фигурными</p>

<pre>+ z = x+y} > a [1] 2</pre>	<p>скобками. Его значение – результат последнего присваивания в блоке.</p>
<pre>> 2 + 3 [1] 5 > "+"(2, 3) # можно и так [1] 5</pre>	<p>Сложение.</p> <p>Любая операция – это прежде всего функция!</p>
<pre>> 7/3 [1] 2.333333 > 7%/%3 [1] 2 > 7%%3 [1] 1 > 2^3 [1] 8</pre>	<p>Деление.</p> <p>Целочисленное деление.</p> <p>Остаток.</p> <p>Возведение в степень.</p>
<pre>> round(1.234, 1) [1] 1.2 > floor(5.2) [1] 5 > ceiling(5.2) [1] 6 > trunc(5.2) [1] 5 > signif(0.55555555, digits = 3) [1] 0.556</pre>	<p>Округление до заданного знака после запятой.</p> <p>Другие функции округления.</p>
<pre>> round(0.5) [1] 0</pre>	<p>Обратите внимание: R использует стандарт ИЕС 60559.</p>
<pre>> 1==2 [1] FALSE > x = F > x [1] FALSE > FALSE & TRUE # и [1] FALSE > FALSE TRUE # или [1] TRUE > !FALSE # не [1] TRUE x <- 2 if ((x>0) && (x<5)) x <- 5</pre>	<p>Булев тип.</p> <p>Допустимы сокращения: «Т» и «F», если, конечно, нет переменных с таким именем.</p> <p>Логические операции: конъюнкция, дизъюнкция, отрицание (определены и над логическими векторами, см. дальше).</p> <p>В логических выражениях используйте && и .</p>
<pre>> c(0, 0, -1, NA) / c(0, 1, 0, 1) [1] NaN 0 -Inf NA > c(0, 0, -1, NA) * c(0, 1, 0, 1) [1] 0 0 0 NA</pre>	<p>Есть не-числа (NaN), бесконечности (Inf, – Inf) и «отсутствия значений» (NA).</p> <p>Деление и умножение поэлементное (те, что приняты в линейной алгебре – %/%, %*%).</p>
<pre>> var(x) [1] 4.666667 > sum((x-mean(x))^2) / (length(x) - 1)</pre>	<p>Вызов функций (два одинаковых действия).</p>

[1] 4.666667	
> sqrt(-1) [1] NaN Warning message: In sqrt(-1) : созданы NaN > sqrt(-1+0i) [1] 0+1i	Для операции с комплексными числами указывайте мнимую часть! Обратите внимание на запись «0i».
> 0.3 - 3*0.1 [1] -5.551115e-17	Напоминаем про особенность вычислений (особенность IEEE 754 арифметики)! Поэтому > 0.3 == 3*0.1 [1] FALSE
> formatC(pi, digits=2, width=8, format="f") [1] " 3.14"	Один из низкоуровневых способов управлять форматом вывода чисел (возвращает строку).
> source("commands.R")	Вызов команд из файла.
> sink("record.lis") > sink()	Перенаправить вывод в файл. Вернуть вывод на консоль.
> objects() [1] "a" "b" > rm(a) > objects() [1] "b"	Вывести список объектов . Есть также функция ls() . Удалить объект.

Векторы

> x <- 1 > y[1] <- 1 Ошибка в y[1] <- 1 : объект 'y' не найден > (y = vector(length=2)) [1] FALSE FALSE > y[1] = 1 > y [1] 1 0	Декларировать переменные заранее не надо. Однако есть исключения. Вектор надо сначала создать (чтобы обращаться по индексам). Названия переменных в R (например, «x») – просто указатели.
> 1:5 [1] 1 2 3 4 5 > seq(from=1, to=5) [1] 1 2 3 4 5 > seq(from=1, to=5, by=2) [1] 1 3 5 > seq(from=1, to=5, length=2) [1] 1 5 > 5:1 [1] 5 4 3 2 1	Генерация векторов. Не надо указывать (-1) как в системе MATLAB.
> a <- rep(c(1,3), times=3) > length(a) <- 5 > a [1] 1 3 1 3 1 # это было обозначение... > a <- "length<-"(a, 5)	Повтор вектора. «Урезание» вектора. Обратите внимание: это «нетипичная» запись. Непривычно делать присваивание значению функции, но на самом деле в R есть функция "length<-". И это было просто её обозначение!
> rep(1:3, each=2)	Повтор каждого элемента вектора.

<pre>[1] 1 1 2 2 3 3 > rep(1:3, 2) # times = 2 [1] 1 2 3 1 2 3 > rep(1:3, times=2, len=7) [1] 1 2 3 1 2 3 1 > rep(1:3, 1:3) [1] 1 2 2 3 3 3</pre>	<p>Повтор каждого элемента нужное число раз.</p>
<pre>> rev(1:4) [1] 4 3 2 1</pre>	<p>«Переворот» вектора. Это замена такого фрагмента: <code>a = 1:4</code> <code>a[length(a):1]</code></p>
<pre>> x = c(2, 1, 10, 9, 10) > sort(x) [1] 1 2 9 10 10 > order(x) [1] 2 1 4 3 5 > sort(x, decreasing = TRUE) [1] 10 10 9 2 1 > rank(c(2, 3, 3, 5, 5, 5, 4)) [1] 1.0 2.5 2.5 6.0 6.0 6.0 4.0</pre>	<p>Сортировка вектора. Сам вектор x при этом не меняется! Индексы элементов в порядке возрастания.</p> <p>Сортировка в другом порядке.</p> <p>Порядковый ранг элементов. Очень полезная функция при обработке данных.</p>
<pre>> c(1, 4, 2, 6) > 4 [1] FALSE FALSE FALSE TRUE > is.na(c(1, NA, 3)) [1] FALSE TRUE FALSE > which(c(T, T, F, T)) [1] 1 2 4</pre>	<p>Логический вектор.</p> <p>Что пропуск. Напомним, NA – пропуск, NAN – не-число.</p> <p>Перевод логического вектора в индексы.</p>
<pre>> a = c(2, -Inf, 5, NaN, NA) # что является числом > i = is.finite(a) > a[i] [1] 2 5</pre>	<p>Логический вектор возвращают многие функции.</p> <p>Логическим вектором можно делать индексацию.</p>
<pre>> x <- c(11, 45, 12, 9) > y <- c(4, 3, 1, 2) > z <- -c(1, 4, 3, 2) > x[y[z]] [1] 9 45 11 12</pre>	<p>Индексация «вектор от вектора».</p> <p>Напоминаем, элементы вектора нумеруются с единицы!</p>
<pre>> x <- c(1, 5, 2, 3, 10) > x[-c(2, 4)] [1] 1 2 10 > (x <- x[-3]) [1] 1 5 3 10 > (x <- c(x[1:2], 13, x[3:4])) [1] 1 5 13 3 10</pre>	<p>Все остальные элементы (кроме второго и четвертого). Очень удобная запись!</p> <p>Так можно удалить элемент из вектора. Только так, поскольку размер вектора определяется при создании!</p> <p>Вставка элемента.</p>
<pre>> fruit <- c(5, 10, 1, 20) > names(fruit) <- c("orange", "banana", "apple", "peach") > fruit["banana"]</pre>	<p>«Именованное индексирование». Можно обращаться «по имени».</p>

<pre>banana 10 > fruit[2] banana 10 # обнуление имён > names(fruit) <- NULL > fruit["banana"] [1] NA > fruit[2] [1] 10</pre>	
<pre>> a <- 1 > a[4] <- 2 > a [1] 1 NA NA 2</pre>	<p>Обращение к элементам вектора. Автоматическое дополнение вектора «неизвестными значениями».</p>
<pre>> a <- numeric(0) > a[2] <- 3 > a [1] NA 3 > mode(a) [1] "numeric" > length(a) [1] 2 > a[3] = 2i > mode(a) [1] "complex" > a[4] = "q" > mode(a) [1] "character" > a [1] NA "3+0i" "0+2i" "q"</pre>	<p>Создание пустого вектора и присваивание.</p> <p>Тип объекта.</p> <p>Длина объекта.</p> <p>Смена типа. Обратите внимание на динамическую смену типа.</p> <p>Ещё одна «смена».</p>
<pre>x <- c() # можно x = NULL for (i in 1:10) x <- append(x, i) x [1] 1 2 3 4 5 6 7 8 9 10</pre>	<p>«Наращивание» вектора.</p>
<pre>> x <- c() > 1:length(x) [1] 1 0 > seq(c(7,2,1)) [1] 1 2 3 > seq(c()) integer(0)</pre>	<p>Тонкость: конструкция <code>1:length(x)</code> не всегда работает «ожидаемо». Выход – использовать <code>seq</code>.</p>
<pre>> identical(c(1,2), c(1,2)) [1] TRUE > identical(c(1,2), c(1,3)) [1] FALSE > identical(c(1,2), 1:2) [1] FALSE</pre>	<p>Есть функция проверки на равенство, но пользоваться ей надо с особой осторожностью.</p>

<pre># вот почему... > typeof(c(1,2)) [1] "double" > typeof(1:2) [1] "integer"</pre>	
<pre>> crossprod(c(1,3,2), c(0,1,2)) [,1] [1,] 7</pre>	Скалярное произведение векторов.
<pre>> a <- 1:6 > b <- c(-2,1) > a + b [1] -1 3 1 5 3 7</pre>	Векторы дублируются при присваивании.

Математические операции

<pre>> A = c(1,2,3,1) > B = c(1,3,4,5) > union(A,B) # объединение [1] 1 2 3 4 5 > setdiff(A,B) # разность [1] 2 > intersect(A,B) # пересечение [1] 1 3 > setequal(A,1:3) # равенство [1] TRUE > 2 %in% A # принадлежность [1] TRUE > choose(3,2) # число сочетаний [1] 3 > combn(1:4,3) # сочетания [,1] [,2] [,3] [,4] [1,] 1 1 1 2 [2,] 2 2 3 3 [3,] 3 4 4 4 > combn(1:4, 3, sum) [1] 6 7 8 9</pre>	<p>Операции на множествах.</p> <p>Различные сочетания без повторений. Результат записывается в матрицу.</p> <p>Есть возможность сразу применить некоторую операцию к столбцам этой матрицы.</p>
<pre>> a = c(3,5,1) > b = c(4,2) > max(a,b) # максимальный эл-т [1] 5 > which.max(a) # его индекс [1] 2 > pmax(a,b) # поэлементный max [1] 4 5 4 Предупреждение In pmax(a, b) : аргумент будет частично зациклен</pre>	<p>Операции максимума. Аналогично с минимумом – надо max заменить на min. Максимальный элемент.</p> <p>Его индекс.</p> <p>Поэлементный максимум. Поскольку у аргументов разные размеры, один из них «зацикливается».</p>
<pre>> nlm(function(x) return(abs(x)+1/abs(x)), 8) \$minimum [1] 2 \$estimate</pre>	<p>Минимизация функции, см. также optim. Обратите внимание на механизм анонимных функций в R.</p>

<pre>[1] 0.9999995 \$gradient [1] 1.776357e-09 \$code [1] 1 \$iterations [1] 13</pre>	
<pre>> cumsum(c(4,2,1)) [1] 4 6 7 > cumprod(c(4,2,1)) [1] 4 8 8</pre>	<p>Кумулятивная сумма.</p> <p>Кумулятивное произведение.</p>
<pre>> factorial(12) [1] 479001600 > factorial(1:4) [1] 1 2 6 24</pre>	Факториал.
<pre>> exp = expression(x*sqrt(x+1)+cos(x)) > D(exp, 'x') sqrt(x + 1) + x * (0.5 * (x + 1)^-0.5) - sin(x)</pre>	<p>Символьные вычисления –</p> <p>взятие производной.</p>
<pre>> integrate(function(x) sin(x)*x, 0, 2*pi) -6.283185 with absolute error < 1.4e-13</pre>	Интегрирование.
<pre># Normal dnorm() pnorm() qnorm() rnorm() # Chi square dchisq() pchisq() qchisq() rchisq() # Binomial dbinom() pbinom() qbinom() rbinom()</pre>	<p>Различные статистические функции для трёх распределений. Легко запомнить их назначения по первой букве</p> <p>d – функция плотности, p – функция распределения, q – квантиль, r – генерация случайных чисел.</p>
<pre>> set.seed(100) > rnorm(3) # (1) [1] -0.502 0.131 -0.078 > rnorm(3) [1] 0.886 0.116 0.318 > set.seed(100) # == > rnorm(3) # повтор (1) [1] -0.502 0.131 -0.078</pre>	Установка генератора случайных чисел.
<pre>> runif(4) [1] 0.342 0.730 0.378 0.774</pre>	Равномерно распределённые значения.

<pre>> rnorm(4) [1] -0.280 -1.008 -1.070 -1.094</pre>	Нормально распределённые значения.
<pre>> sample(1:10, 3, replace=FALSE) [1] 3 10 6 > s = c('A', 'B', 'C', 'D') > sample(s, 5, replace=TRUE) [1] "A" "D" "D" "A" "A" > sample(0:1, 10, replace=TRUE, p=c(0.3, 0.7)) [1] 1 0 1 1 1 1 0 1 1 1</pre>	<p>3 случайных числа из 10 без повторений.</p> <p>5 случайных букв.</p> <p>Случайный бинарный вектор (с заданными вероятностями появления 0 и 1).</p>
<pre>> a = c(20, 10, 40, 30, 30) > sort(a) [1] 10 20 30 30 40 > order(a) [1] 2 1 4 5 3 > rank(a) [1] 2.0 1.0 5.0 3.5 3.5</pre>	<p>Сортировка.</p> <p>order – даёт индексы элементов в порядке возрастания, для убывания используйте аргумент decreasing = FALSE.</p> <p>rank – аналогичная функция, которая учитывает одинаковость элементов.</p>

Матрицы

<pre>a = matrix(1:6, nrow=2, ncol=3) [,1] [,2] [,3] [1,] 1 3 5 [2,] 2 4 6 a = matrix(1:6, nr=2) [,1] [,2] [,3] [1,] 1 3 5 [2,] 2 4 6 > a[1,2] [1] 3 a[,2] [1] 3 4 a[,2, drop=FALSE] [,1] [1,] 3 [2,] 4 > a[, 2:3] [,1] [,2] [1,] 3 2 [2,] 1 3 > a[3] [1] 3 c(a) [1] 1 2 3 4 5 6 > a+c(1,2,3) [,1] [,2] [,3] [1,] 2 6 7 [2,] 4 5 9</pre>	<p>Порождение матрицы.</p> <p>Аналогичное действие (один аргумент не указывается, а второй – указывается в «сокращённом варианте»).</p> <p>Индексация (как и для векторов – нумерация с единицы). Второй столбец.</p> <p>Второй столбец как подматрица.</p> <p>Подматрица.</p> <p>Третий элемент (если считать по столбцам).</p> <p>Все элементы матрицы.</p> <p>Обратите внимание, что операция выполнялась по столбцам! Матрицы также хранятся по столбцам (это просто «длинные векторы», которые «имею форму»)! Напоминаем, также, что векторы дублируются при присваивании.</p>
--	---

<pre>> rownames(a) = c("X", "Y") > colnames(a) = c("Z", "T", "S") > a["X", "T"] [1] 3 > a Z T S X 1 3 2 Y 2 1 3</pre>	<p>Именованние строк и столбцов.</p>
<pre>> ncol(a) [1] 3 > nrow(a) [1] 2 > dim(a) [1] 2 3 > dim(a)[1] [1] 2</pre>	<p>Число столбцов.</p> <p>Число строк.</p> <p>Размеры матрицы.</p> <p>Второй способ – число строк. Обратите внимание на синтаксис.</p>
<pre>> a <- matrix(1:4, nr=2) > ncol(a) [1] 2 > b <- a[,2] > ncol(b) # над вектором! NULL > b [1] 3 4 > b <- a[,2, drop=FALSE] > ncol(b) # над матрицей! [1] 1 > b [,1] [1,] 3 [2,] 4 # [– это функция > b <- "["(a, 1:2, 2, drop=FALSE) > b [,1] [1,] 3 [2,] 4 > as.matrix(c(1,2)) [,1] [1,] 1 [2,] 2</pre>	<p>Ещё раз обратим внимание, что R часто автоматически «схлопывает» данные в вектор, а это уже не матрица! Поэтому некоторые функции над ним не имеют смысла.</p> <p>Запись <code>a[,2, drop=FALSE]</code> может показаться странной, но квадратная скобка «[» – это обычная функция!</p> <p>Вот ещё один способ превратить вектор в матрицу.</p>
<pre>> a = matrix(1:3, nrow = 2, ncol = 3, byrow=TRUE) > a [,1] [,2] [,3] [1,] 1 2 3 [2,] 1 2 3 > a %*% c(1,1,1) [,1] [1,] 6 [2,] 6 > y <- matrix(nrow=2, ncol=2)</pre>	<p>Порождение матрицы «по строкам». Матрица всё равно будет храниться «по столбцам»!</p> <p>Обратите внимание на умножение! Для вектора нет понятие «ориентации» (точнее, часто считается, что он вертикально ориентирован – попробуйте <code>t(c(1,1,1))</code>).</p> <p>Порождение «по умолчанию».</p>

<pre>> y [,1] [,2] [1,] NA NA [2,] NA NA > y[,1] <- 2 > y[,2] <- c(1,3) > y [,1] [,2] [1,] 2 1 [2,] 2 3</pre>	<p>Присваивание значений элементам.</p>
<pre>> h = 1:10 > attr(h, "dim") = c(2,5) > h [,1] [,2] [,3] [,4] [,5] [1,] 1 3 5 7 9 [2,] 2 4 6 8 10</pre>	<p>Ещё один способ превращения вектора в матрицу. Снова «нетипичная» запись!</p>
<pre>a <- matrix(1:6,2,3) row(a) [,1] [,2] [,3] [1,] 1 1 1 [2,] 2 2 2 col(a) [,1] [,2] [,3] [1,] 1 2 3 [2,] 1 2 3 > a[col(a)==row(a)] <- 0 > a [,1] [,2] [,3] [1,] 0 3 5 [2,] 2 0 6</pre>	<p>Интересные функции «номера строк» и «номера столбцов».</p> <p>Обнуление главной диагонали.</p>
<pre>> cbind(0, rbind(c(1,3), c(5,7))) [,1] [,2] [,3] [1,] 0 1 3 [2,] 0 5 7 > rbind(c(1,2), c(3,4)) [,1] [,2] [1,] 1 2 [2,] 3 4 > cbind(c(1,2), c(3,4)) [,1] [,2] [1,] 1 3 [2,] 2 4</pre>	<p>Конкатенация матриц.</p> <p>Так удобно порождать матрицы. Напоминаем, про отсутствие ориентации у вектора.</p>

Операции с матрицами

<pre>> A <- matrix(1:4, nr=2) > B <- matrix(-2:1, nr=2) > A %*% B [,1] [,2] [1,] -5 3 [2,] -8 4 > A + B</pre>	<p>Обычное матричное умножение.</p> <p>Сложение.</p>
---	--

<pre> [,1] [,2] [1,] -1 3 [2,] 1 5 > A + 1 [,1] [,2] [1,] 2 4 [2,] 3 5 > A*B [,1] [,2] [1,] -2 0 [2,] -2 4 > c(A) [1] 1 2 3 4 > A[A<=2] <- 0 > A [,1] [,2] [1,] 0 3 [2,] 0 4 > A[A>2] [1] 3 4 > which(A>2) [1] 3 4 </pre>	<p>Прибавление константы (происходит поэлементно).</p> <p>Поэлементное умножение.</p> <p>Векторизация. Кстати, операции A+B, c(A) +B, A+c(B) эквивалентны.</p> <p>Фильтрация.</p> <p>Элементы, удовлетворяющие условию. Результат, естественно, выводится как вектор!</p> <p>Их номера. Обратите внимание: номера соответствуют номерам в векторе</p>
<pre> > outer(c(1,2,3), c(-1,2,1), "*") [,1] [,2] [,3] [1,] -1 2 1 [2,] -2 4 2 [3,] -3 6 3 > c(1,2,3)%o%c(-1,2,1) [,1] [,2] [,3] [1,] -1 2 1 [2,] -2 4 2 [3,] -3 6 3 </pre>	<p>«Внешние» (покомпонентные) операции. Способ умножить вектор-столбец на вектор-строку.</p> <p>Кстати, для скалярного произведения в R нет функции «inner», есть функция crossprod</p>
<pre> > A = matrix(c(1,0,2,-1), nrow=2) > B = matrix(c(0,-1,3,2), nrow=2) > kronecker(A, B) [,1] [,2] [,3] [,4] [1,] 0 3 0 6 [2,] -1 2 -2 4 [3,] 0 0 0 -3 [4,] 0 0 1 -2 </pre>	<p>Кroneckerовское произведение.</p>
<pre> > A <- matrix(1:4, ncol = 2) > A = t(A) > A [,1] [,2] [1,] 1 2 [2,] 3 4 </pre>	<p>Транспонирование.</p>

<pre>> b = c(5,7) > solve(A,b) [1] -3 4 > det(A) [1] -2 > solve(A) [,1] [,2] [1,] -2.0 1.0 [2,] 1.5 -0.5</pre>	<p>Решение уравнений.</p> <p>Определитель.</p> <p>Обратная матрица. Внимание: \mathbf{A}^{-1} – не обратная матрица (здесь степень поэлементная).</p>
<pre>> eigen(A) \$values [1] 5.3722813 -0.3722813 \$vectors [,1] [,2] [1,] -0.4159736 -0.8245648 [2,] -0.9093767 0.5657675</pre>	<p>Собственные значения и векторы.</p>
<pre>> svd(a) \$d [1] 5.291503e+00 2.943727e-16 \$u [,1] [,2] [1,] -0.7071068 -0.7071068 [2,] -0.7071068 0.7071068 \$v [,1] [,2] [1,] -0.2672612 -0.9487015 [2,] -0.5345225 0.2918172 [3,] -0.8017837 0.1216890 > svd(a)\$d [1] 5.291503e+00 2.943727e-16</pre>	<p>Сингулярное разложение матрицы.</p> <p>Только одна матрица разложения.</p>
<pre>> qr(a) \$qr [,1] [,2] [1,] -3.1622777 -4.4271887 [2,] 0.9486833 -0.6324555 \$rank [1] 2 \$graux [1] 1.3162278 0.6324555 \$pivot [1] 1 2 attr("class") [1] "qr"</pre>	<p>QR-разложение, есть также разложение chol.</p>
<pre>> f <- function(x) # нормировка</pre>	<p>Пример как делать нормировки (по строкам и</p>

<pre>+ {x/sum(x)} > A <- matrix(1:6, nc=3) > A [,1] [,2] [,3] [1,] 1 3 5 [2,] 2 4 6 > t(apply(A, 1, f)) # по строкам [,1] [,2] [1,3] [1,] 0.1111 0.3333 0.5555 [2,] 0.1666 0.3333 0.5000 > apply(A, 2, f) # по столбцам [,1] [,2] [1,3] [1,] 0.3333 0.4285 0.4545 [2,] 0.6666 0.5714 0.5454</pre>	<p>столбцам) – с помощью очень универсальной функции apply.</p>
<pre>> A = matrix(sample(10, 6), nr=2, nc=3) > A [,1] [,2] [,3] [1,] 6 9 3 [2,] 1 2 4 > which.min(A) [1] 2 > apply(A, 1, which.min) [1] 3 1 > apply(A, 2, which.min) [1] 2 2 1</pre>	<p>Ещё раз об apply... многие функции в R работают сразу со всеми элементами матрицы. Здесь показано, как найти номера наименьших элементов в строках и столбцах.</p> <p>См. также Векторизация, функции apply-семейства.</p>
<pre>> m <- matrix(1:4, nr=2) > m [,1] [,2] [1,] 1 3 [2,] 2 4 > sweep(m, 1, c(1,2), "+") [,1] [,2] [1,] 2 4 [2,] 4 6 > sweep(m, 2, c(1,2), "+") [,1] [,2] [1,] 2 5 [2,] 3 6 > sweep(m, 2, colSums(m), "/") [,1] [,2] [1,] 0.3333333 0.4285714 [2,] 0.6666667 0.5714286</pre>	<p>Функция sweep позволяет делать операции по строкам и по столбцам.</p> <p>Прибавить вектор к каждому столбцу.</p> <p>К каждой строке.</p> <p>Нормировка – делим на сумму элементов в столбце.</p> <p>См. также Векторизация.</p>
<pre>> m <- matrix(1:6, nr=2) > m [,1] [,2] [,3] [1,] 1 3 5 [2,] 2 4 6 > diag(m) [1] 1 4</pre>	<p>Функция diag может делать разные операции, в зависимости от аргумента: возвращать диагональ матрицы или порождать матрицу с заданной диагональю.</p> <p>В последнем случае можно указать размеры, например так: diag(c(1,4), 2, 3).</p>

<pre>> diag(c(1,4))</pre> <pre> [,1] [,2]</pre> <pre>[1,] 1 0</pre> <pre>[2,] 0 4</pre>	
---	--

Многомерные матрицы

<pre>> a = array(1:12, c(2,3,2))</pre> <pre>> a</pre> <pre>, , 1</pre> <pre> [,1] [,2] [,3]</pre> <pre>[1,] 1 3 5</pre> <pre>[2,] 2 4 6</pre> <pre>, , 2</pre> <pre> [,1] [,2] [,3]</pre> <pre>[1,] 7 9 11</pre> <pre>[2,] 8 10 12</pre> <pre>> dim1 = c("A","B")</pre> <pre>> dim2 = c("A2","B2","C2")</pre> <pre>> dim3 = c("X","Y")</pre> <pre>> dimnames(a) =</pre> <pre>list(dim1,dim2,dim3)</pre> <pre>> a["A","B2","Y"]</pre> <pre>[1] 9</pre>	Многомерные матрицы.
<pre>> A <- matrix(1:6, nr=2)</pre> <pre>> B <- matrix(7:12, nr=2)</pre> <pre>> array(c(A,B), dim=c(2,3,2))</pre> <pre>, , 1</pre> <pre> [,1] [,2] [,3]</pre> <pre>[1,] 1 3 5</pre> <pre>[2,] 2 4 6</pre> <pre>, , 2</pre> <pre> [,1] [,2] [,3]</pre> <pre>[1,] 7 9 11</pre> <pre>[2,] 8 10 12</pre>	Конкатенация матриц по третьему направлению (по третьей размерности).
<pre>> a = 1:24</pre> <pre>> dim(a) = c(4,3,2)</pre> <pre>> a</pre> <pre>, , 1</pre> <pre> [,1] [,2] [,3]</pre> <pre>[1,] 1 5 9</pre> <pre>[2,] 2 6 10</pre> <pre>[3,] 3 7 11</pre> <pre>[4,] 4 8 12</pre>	Ещё один способ создания многомерной матрицы.

	[,]	[,]	[,]	
[1,]	13	17	21	
[2,]	14	18	22	
[3,]	15	19	23	
[4,]	16	20	24	

Списки

<pre> > l <- list(name="John", salary=100, credit=TRUE) > str(l) # созданный список List of 3 \$ name : chr "John" \$ salary: num 100 \$ credit: logi TRUE > l[["name"]] # по имени \$name [1] "John" > l\$name # по имени (2й способ) [1] "John" > l[[1]] # по индексу [1] "John" > l[1] # Одинарные [] - подсписок \$name [1] "John" > l[1:2] # подсписок \$name [1] "John" \$salary [1] 100 > l[[1:2]] # так нельзя Ошибка в l[[1:2]] : подгруппа выходит за пределы > length(l) # длина [1] 3 </pre>	<p>Список – перечень разнотипных данных. Похож на питоновский словарь и очень отдалённо на структуру в С.</p> <p>Для вывода списка лучше использовать str – получается более компактно. «str» означает «структура» (функция вызывается от любого объекта).</p> <p>Индексация возможна по имени и по номеру индекса (начинается с единицы).</p> <p>Обратите внимание на R-й способ индексации через «\$».</p> <p>Когда используем одинарные квадратные скобки, результатом является список – подсписок исходного.</p> <p>Длина списка.</p>
<pre> > l <- vector(mode="list") > l list() > l[1] = 1 > l["second"] = 2 </pre>	<p>Создание пустого списка.</p> <p>«Наполнение» списка.</p>

<pre>> str(l) List of 2 \$: num 1 \$ second: num 2 > names(l)[1] = "first" > str(l) List of 2 \$ first : num 1 \$ second: num 2</pre>	<p>У первого элемента отсутствует имя – обращаться можно только по индексу. Добавление имени.</p>
<pre>> a <- matrix(c(3,5,2,1), nrow=2) > b <- "12 3" > l <- list(a,b,12) > l[[1]][2][1] [1] 5 > names(l) <- c("first", "second", "third") > index = 2 > l[[index]] [1] "12 3" > l\$index # так нельзя NULL > index <- "second" > l[[index]] [1] "12 3" > l\$index # так нельзя NULL > l\$onemore <- TRUE # добавить > l\$second <- NULL # удалить > str(l) List of 3 \$ first : num [1:2, 1:2] 3 5 2 1 \$ third : num 12 \$ onemore: logi TRUE > str(unname(l)) List of 3 \$: num [1:2, 1:2] 3 5 2 1 \$: num 12 \$: logi TRUE > names(l) <- NULL</pre>	<p>Обращение к (2,1)-элементу матрицы, которая является первым элементом списка.</p> <p>Ещё раз про различные способы индексации...</p> <p>Добавление и удаление элементов списка.</p> <p>Первый способ «отказаться от имён» – функция unname.</p> <p>Второй – «обнуление» имён.</p>
<pre>> l <- hist(c(1,1,2,2,3,2)) > str(l) List of 7 \$ breaks : num [1:5] 1 1.5 2 2.5 3 \$ counts : int [1:4] 2 3 0 1</pre>	<p>Многие функции возвращают списки. Это способ вернуть сразу несколько значений (формально возвращая одно).</p>

<pre> \$ intensities: num [1:4] 0.667 1 0 0.333 \$ density : num [1:4] 0.667 1 0 0.333 \$ mids : num [1:4] 1.25 1.75 2.25 2.75 \$ xname : chr "c(1, 1, 2, 2, 3, 2)" \$ equidist : logi TRUE - attr(*, "class")= chr "histogram" > summary(1) breaks Length Class Mode numeric 5 -none- counts 4 -none- numeric intensities 4 -none- numeric density 4 -none- numeric mids 4 -none- numeric xname 1 -none- character equidist 1 -none- logical </pre>	<p>Ещё один способ вывода информации о списке. Попробуйте также plot(1) – это ещё один пример т.н. generic-функции, которую можно вызывать от любого объекта.</p>
<pre> > c(1,c(2,3)) # вектор [1] 1 2 3 > c(1,list(2,3)) # список [[1]] [1] 1 [[2]] [1] 2 [[3]] [1] 3 # СПИСОК СПИСКОВ > list(1,list(2,3)) [[1]] [1] 1 [[2]] [[2]][[1]] [1] 2 [[2]][[2]] [1] 3 </pre>	<p>Тонкость: функция c не получает «двухуровневые» объекты. Она всё «сливает» в один вектор/список.</p> <p>А функция list получает!</p>
<pre> > a <-list(1, list(2, 3)) </pre>	<p>Векторизация списков.</p>

<pre>> names(a) <-c('a1', 'a2') > names(a\$a2) <-c('a1', 'a2') > c(a) # не получится... \$a1 [1] 1 \$a2 \$a2\$a1 [1] 2 \$a2\$a2 [1] 3 > c(a, recursive=TRUE) # вектор! a1 a2.a1 a2.a2 1 2 3</pre>	<p>Нужно указывать значение аргумента recursive.</p>
<pre>> l <- list(1,1:2,1:3,1:4) > f <-function(x) return(c(min(x),max(x))) > lapply(l, f) # результат - список [[1]] [1] 1 1 [[2]] [1] 1 2 [[3]] [1] 1 3 [[4]] [1] 1 4 > sapply(l, f) # результат - матрица [,1] [,2] [,3] [,4] [1,] 1 1 1 1 [2,] 1 2 3 4</pre>	<p>Применение apply-функций к списку. Для каждого элемента списка (это векторы) находим максимальное и минимальное значения.</p>

Циклы и функции

<pre>> a = 2; > if (a<3) a = a+5 else a = a-7 > a [1] 7</pre>	<p>Условный оператор. Скобки важны!</p>
<pre>> a=3 > x = if(a>2) 5 else 7 > x [1] 5 > ifelse(a>2, 5, 7) [1] 5 > switch(a, 'one', 'two', 'three')</pre>	<p>Можно присваивать значение условного оператора.</p> <p>Удобная конструкция ifelse. Но есть тонкий момент: попробуйте ifelse(2>1, c(7, 2), 3)</p> <p>switch</p>

<pre>[1] "three" > x = c(1, 2, 0, 1, 2, 3) > ifelse(x>1.5, NA, x) [1] 1 NA 0 1 NA NA</pre>	<p>Использование ifelse. Замена «больших» чисел на NA.</p>
<pre>> suma = 0 > a = c(2, 6, 3) # что суммировать > for (i in a) > suma = suma + i</pre>	<p>Оператор цикла for.</p> <p>Конечно, надо делать так: suma = sum(a).</p>
<pre>while(...) { ... } i=0 repeat { i = i+1 if (i<3) next # след. итер. цикла i = i+2 if (i>10) break # закончить print(i) }</pre>	<p>Синтаксис циклов while и repeat.</p> <p>Обратите внимание: next, а не continue.</p>
<pre>> myfun <- function(a, b, i=TRUE) + { if (i) + return (a+b) + else + return(a-b) + } > myfun(1, 3) [1] 4 > myfun(1, 3, FALSE) [1] -2 > myfun(1, i=FALSE, b=3) [1] -2 > f <- myfun > f(0, 1, 1==2) [1] -1</pre>	<p>Функции. Значение функции – последнее вычисленное или return.</p> <p>Важно понимать, что function – встроенная функция, «работа которой» – создавать другие функции.</p> <p>Можно придавать аргументам значения по умолчанию (i=TRUE).</p> <p>Возможны обращения при явном указании аргументов.</p> <p>Функция – это объект. Можно выполнять присваивание.</p>
<pre>myfun <- function(a, b) { c = a+b d = a-b return(list(c,d)) } > myfun(1, 2) [[1]] [1] 3 [[2]] [1] -1 > myfun</pre>	<p>Несколько возвращаемых значений.</p> <p>Такой способ принят во многих языках программирования. Переменные c, d могут иметь разный тип!</p> <p>Вызов функции.</p> <p>Если набрать имя объекта, то мы увидим</p>

<pre>function(a, b) { c = a+b d = a-b return(list(c,d)) } > formals(myfun) \$a \$b > body(myfun) { c = a + b d = a - b return(list(c, d)) } > body(myfun) <- quote(max(a, 1) + min(b, 0))</pre>	<p>объект. Функции не являются исключением.</p> <p>Специальный вызов аргументов.</p> <p>Специальный вызов «тела функции».</p> <p>Кстати, для редактирования функции наберите <code>edit(myfun)</code>.</p> <p>«Подмена» тела функции. Если не использовать <code>quote</code>, то R начнёт вычислять выражение <code>max(a, 1) + min(b, 0)</code>.</p>
<pre>> f <- function(...) {list(...)} > f(1, 2, 3) [[1]] [1] 1 [[2]] [1] 2 [[3]] [1] 3</pre>	<p>Произвольное число аргументов у функции.</p>
<pre>f <- function (...) { l <- list(...) for (i in seq(along=l)) { cat("Имя:", names(l)[i], "Значение:", l[[i]], "\n") } } > f(x=1, g=3) Имя: x Значение: 1 Имя: g Значение: 3</pre>	<p>«Прохождение» по списку аргументов.</p>
<pre>f <- function(a) { b <- 1 x <- ls() c <- 2 return(x) } f(0) [1] "a" "b"</pre>	<p>Вывод всех объектов (имён) в локальной области осуществляется с помощью функции <code>ls</code>. Заметим, что имя переменной <code>c</code> не выводится (она ещё не создана).</p>

<pre>f <- function(xloc) {return(xloc+xglob)} xglob = 10 > f(2) [1] 12</pre>	<p>Допустимо использовать глобальные переменные.</p>
<pre>f <- function(x) { x <- x+1 # глобально! y <- 2 # идёт только в ответ } z <- 3 > (f(z)) [1] 2 > x [1] 4 > y Ошибка: объект 'y' не найден > z [1] 3</pre>	<p>В этом примере: переменная x после вызова функции существует. Она стала глобальной, поскольку использовалось присваивание <-, переменная y, напротив, не существует... но она была результатом работы функции.</p> <p>Присваивание <- ищет нужную переменную в окружении предыдущего уровня, потом предпредыдущего и т.д. Если не находит – создаёт глобальную переменную.</p>
<pre>> predict function (object, ...) UseMethod("predict") <environment: namespace:stats> > methods("predict") [1] predict.ar* predict.Arima* predict.arima0* predict.glm predict.HoltWinters* predict.lm [7] predict.loess* predict.mlm predict.nls* predict.poly predict.ppr* predict.prcomp* [13] predict.princomp* predict.smooth.spline* predict.smooth.spline.fit* predict.StructTS*</pre>	<p>Просмотр кода может не работать (функция predict определена для разных объектов) – надо уточнить название функции – сработает predict.mlm</p>
<pre>> "%!%" <- function(a, b) {a+b+min(a,b)} > 2%!%3 [1] 7</pre>	<p>Определение новой операции.</p>
<pre>> f <- function (x) { if(!is.numeric(x)) + stop("Expecting a NUMERIC vector!") + x^2 + }</pre>	<p>Остановка функции.</p>

<pre>> f('as') Ошибка в f("as") : Expecting a NUMERIC vector! > f(1:2) [1] 1 4</pre>	
<pre>my.arg <- function (x, ...) { cat("My first argument was: ") cat(deparse(substitute(x))) cat("\n") } > my.arg(3) My first argument was: 3 > my.arg(x) My first argument was: x > my.arg(x+1) My first argument was: x + 1</pre>	<p>Функция «видит свои аргументы».</p>
<pre>> counter <- function() + { + n <- 0 # инициализация + function() + { + n<-n+1 # увелич. + cat('Счётчик=', n, '\n') + } + } > n1 <- counter() > n2 <- counter() > n1 function() { n<-n+1 cat('Счётчик=', n, '\n') } <environment: 0x07312da4> > n2 function() { n<-n+1 cat('Счётчик=', n, '\n') } <environment: 0x0400a870> > n1() Счётчик= 1 > n1() Счётчик= 2 > n2() Счётчик= 1 > n1() Счётчик= 3</pre>	<p>Окружения позволяют создавать подобные функции-счётчики.</p>

Векторизация

– один из инструментов, позволяющих избежать циклов

<pre>> x <- c(1,0,2) > sin(x) [1] 0.84147 0.00000 0.90929 > y <- c(2,0,1) > x>y [1] FALSE FALSE TRUE</pre>	Основной принцип: вектор на вход, вектор на выход.
<pre>> a = 1:4; b = c(0,2) > a + b # нет предупреждения! [1] 1 4 3 6 > a + c(2,0,1) [1] 3 2 4 6</pre> <p>Предупреждение In a + c(2, 0, 1) : длина большего объекта не является произведением длины меньшего объекта</p>	R по умолчанию всё «зацикливает». Иногда даже без предупреждения.
<pre>> x <- c(2,4,NA,3,7,1,4) > x[x>3] [1] 4 NA 7 4 > subset(x, x>3) [1] 4 7 4 > which(x>3) [1] 2 5 7</pre>	Использование логических массивов для фильтрации. Фильтрация с помощью subset . Отличие в интерпретации NA. Индексы нужных элементов.
<pre>> x <- c(1,2,-1,0,3,-2) > ifelse(x>0, x*2, 0) [1] 2 4 0 0 6 0</pre>	Векторизация с помощью ifelse . Здесь, замена отрицательных значений на ноль и удвоение положительных.
<pre>> m <- matrix(1:4, nr=2) > sweep(m, 2, colSums(m), "/") [,1] [,2] [1,] 0.3333333 0.4285714 [2,] 0.6666667 0.5714286</pre>	Использование специальных функций, например sweep . Здесь каждый элемент разделили на сумму элементов в его столбце.

Векторизация, функции apply-семейства

<pre>> x <- 1:5 > f <-function(x) + { + return(c(x, x^2, x^3)) + } > sapply(x, f) [,1] [,2] [,3] [,4] [,5] [1,] 1 2 3 4 5 [2,] 1 4 9 16 25 [3,] 1 8 27 64 125</pre>	Применение функции к каждому элементу вектора. На выходе – матрица. Можно совсем коротко: sapply(1:5, function(x) {c(x, x^2, x^3)})
<pre>> (a <-matrix(sample(12,12), nc=4)) [,1] [,2] [,3] [,4] [1,] 5 1 12 10 [2,] 3 7 8 9 [3,] 2 11 4 6 > apply(a, 1, min) [1] 1 3 2</pre>	Пример построчной/постолбцовой обработки матрицы – всё делается с помощью функции apply . Работает и с многомерными матрицами! Указываем измерение и функцию, которую

<pre>> apply(a, 2, max) [1] 5 11 12 10 > apply(a, 1, sum) [1] 28 27 23 > rowSums(a) [1] 28 27 23</pre>	<p>применить. Здесь: минимальные элементы в строке, максимальные в столбце, суммы строк.</p> <p>Для сумм и средних значений рекомендуется применять функции <code>colSums()</code>, <code>rowSums()</code>, <code>colMeans()</code> и <code>rowMeans()</code>.</p>
<pre>> l <- list(1:3, 'abc', c(3, 4)) > lapply(l, length) [[1]] [1] 3 [[2]] [1] 1 [[3]] [1] 2 > sapply(l, length) [1] 3 1 2 > f <- function(x) + {return(c(min(x), max(x)))} > sapply(l, f) [,1] [,2] [,3] [1,] "1" "abc" "3" [2,] "3" "abc" "4" > sapply(1:2, function(x) matrix(x, 2, 2), simplify = "array") , , 1 [,1] [,2] [1,] 1 1 [2,] 1 1 , , 2 [,1] [,2] [1,] 2 2 [2,] 2 2</pre>	<p>Обрабатываем список и результат получаем в виде списка.</p> <p>А здесь – результат в виде вектора. «s» = «simplify».</p> <p>Иногда (если выводятся векторы одной длины) результат может конвертироваться и в матрицу.</p> <p>И даже в многомерную матрицу.</p>
<pre>> (a <- vapply(NULL, is.factor, FUN.VALUE = FALSE)) logical(0)</pre>	<p>Функция vapply быстрее sapply, но для неё надо указать возвращаемый тип. Рекомендуется использовать именно эту функцию.</p> <p>[vapply – примеры]</p> <p>«v» = «velocity»</p>
<pre>> mapply(sum, 1:4, 0:3, c(10, 10, 0, 0)) [1] 11 13 5 7</pre>	<p>Когда надо поэлементно применить функцию к нескольким объектам, используется mapply. Здесь – сумма первых элементов векторов, вторых и т.д.</p>

	«m» = «multivariate»
<pre>> l <- list(1, c(2,3), list(4, c(5,6))) > f <- function(x) + {x[1]<-x[1]+1; return(x)} > rapply(l, f) [1] 2 3 3 5 6 6 > rapply(l, f, how='replace') [[1]] [1] 2 [[2]] [1] 3 3 [[3]] [[3]][[1]] [1] 5 [[3]][[2]] [1] 6 6</pre>	<p>Рекурсивное применение функции (к вложенным спискам) делается с помощью rapply.</p> <p>Здесь результат в виде списка (перечня значений).</p> <p>А здесь оставляем исходную структуру вложенности!</p> <p>«r» = «recursively»</p>
<pre>> data.frame(name=c("Иванов", "Иванов", "Петров", "Иванов", "Петров", "Сидоров"), mark=c(4,5,3,4,5,5)) name mark 1 Иванов 4 2 Иванов 5 3 Петров 3 4 Иванов 4 5 Петров 5 6 Сидоров 5 > df <- data.frame(name=c("Иванов", "Иванов", "Петров", "Иванов", "Петров", "Сидоров"), mark=c(4,5,3,4,5,5)) > tapply(X = df\$mark, INDEX = df\$name, FUN = mean) Иванов Петров Сидоров 4.333333 4.000000 5.000000</pre>	<p>Пример вычисления средних баллов студентов.</p>

пакет plyr, reshape

Строки

<pre>> 'String'=="String" [1] TRUE</pre>	<p>Строки в R. Можно использовать одинарные или двойные кавычки.</p>
<pre>print('a\tb\n') [1] "a\tb\n" print('a\tb\n', quote=F) [1] a\tb\n</pre>	<p>Выводить строки на экран лучше с помощью cat.</p>

<pre>cat('a\tb\n') a b cat('c:\\temp\\') c:\temp\ cat("в файл", file="filename.txt", append=TRUE)</pre>	<p>Обратите внимание, как cat выводит спецсимволы \t, \n.</p> <p>Запись в файл!</p>
<pre>sprintf('%g-\n%2.2f', 1, 2) [1] "1-\n2.00" strs = c('one', 'two', 'none') > sprintf('q=%s',strs) [1] "q=one" "q=two" "q=none"</pre>	<p>Самое мощное средство для формирования строк – sprintf!</p>
<pre>> x = 13 > paste('x=', x) [1] "x= 13" > paste('x=', x, sep=' ') [1] "x=13" > paste('one', 'two', sep=', ') [1] "one,two"</pre>	<p>Важно использовать sep при конкатенации.</p>
<pre>> a = paste(c("X", "Y"), 1:3, sep="-") > a [1] "X-1" "Y-2" "X-3"</pre>	<p>Специальная конкатенация.</p> <p>Удобно, например, при составлении имён файлов для записи.</p>
<pre>> cat(paste(1:3, "String", sep="-", collapse="\n"), "END\n") 1-String 2-String 3-String END</pre>	
<pre>strs = c('one', 'two', 'none') grep('on',strs) [1] 1 3</pre>	<p>Поиск.</p>
<pre>> s='one, two, three' > nchar(s) [1] 15 > substring(s,6,8) [1] "two" > a = strsplit(s, ',') > a [[1]] [1] "one" " two" " three" > a[[1]][2] [1] " two" > grep('o',s) [1] 1</pre>	<p>Функции со строками.</p> <p>Число символов.</p> <p>Подстрока.</p> <p>Разбиение.</p> <p>Поиск.</p> <p>Тоже поиск.</p>

<pre>> regexpr('o', s) [1] 1 attr(,"match.length") [1] 1 > regexpr('o', c('one','two')) [1] 1 3 attr(,"match.length") [1] 1 1 > gsub(" ", "", s) [1] "one,two,three" > sub(" ", "", s) [1] "one,two, three" > gsub(" +", " ", "a b c d") [1] "a b c d"</pre>	<p>Замена подстрок – здесь – удаление пробелов. Только с первым вхождением. Замена нескольких пробелов одним.</p>
<pre>> grep('[0-9]a', c('12bc', 'q2a', '3', '0aa')) [1] 2 4 > grep('\\.', c('1.2', 'ss', 'ss.', 'q!')) [1] 1 3</pre>	<p>Некоторые применения регулярных выражений. Поиск комбинации «цифра» – буква «а».</p> <p>Поиск точки.</p>

Даты

<pre>> as.Date("2011-12-31") [1] "2011-12-31" > as.Date("31/12/2011", format="%d/%m/%Y") [1] "2011-12-31"</pre>	<p>Конвертирование строки в формат «дата».</p>
<pre>> d = Sys.Date() > d [1] "2012-01-13" > format(d, format="%A, %d %B %Y") [1] "пятница, 13 Январь 2012"</pre>	<p>Из переменной типа «дата» можно выудить достаточно много информации.</p>
<pre>> seq(as.Date("2011-01-01"), as.Date("2011-03-01"), by="2 weeks") [1] "2011-01-01" "2011-01-15" "2011-01-29" "2011-02-12" "2011-02-26"</pre>	<p>Перебор дат с шагом «две недели».</p>
<pre>> seq(as.Date("2012-01-01"), as.Date("2012-03-01"), by="month") [1] "2012-01-01" "2012-02-01" "2012-03-01" > seq(as.Date("2012-01-01"), as.Date("2012-03-01"), by="31 days") [1] "2012-01-01" "2012-02-01"</pre>	<p>Умный перебор дат с шагом «1 месяц» (а не 31 день).</p>
<pre>> a = Sys.time()</pre>	<p>Разница двух засечек.</p>

<pre>> difftime(a, Sys.time(), units="secs") Time difference of -17.214 secs</pre>	
<pre>d <- read.table("foo.txt") d\$Date <- as.Date(as.character(d\$Date))</pre>	Действия при загрузке файла с «временным» столбцом.

Факторы

<pre>> anss = c("Yes", "No", "Yes", "Yes", "No", "I don'n know") > fc <- factor(anss) > fc [1] Yes No Yes Yes No I don'n know Levels: I don'n know No Yes > str(fc) Factor w/ 3 levels "I don'n know",...: 3 2 3 3 2 1 > unclass(fc) [1] 3 2 3 3 2 1 attr("levels") [1] "I don'n know" "No" "Yes" > tapply(c(1,2,2,2,3,4), fc, mean) I don'n know No Yes 4.0000 2.5000 1.6666 > tapply(c(1,2,2,2,3,4), fc, length) I don'n know No Yes 1 2 3 > table(fc) fc I don'n know No Yes 1 2 3</pre>	<p>Факторы. Для хранения «категориальных» («номинальных», факторных) данных (например, национальность, страна проживания, политическая партия, группа крови и т.п.).</p> <p>Видно, что фактически роль играют только номера классов эквивалентностей, порождённых исходными данными.</p> <p>Вычислить средние значения по каждому классу эквивалентности.</p> <p>Мощности классов эквивалентности. Функция tapply наиболее часто используется с факторами.</p> <p>Аналогичный результат – функция table.</p>
<pre>> df <- data.frame(a=c(2,1,2,4), b=c(4,4,2,1)) > df a b 1 2 4 2 1 4 3 2 2 4 4 1 > df[,1] <-factor(df[,1]) > df[,2] <-factor(df[,2]) > df a b 1 2 4 2 1 4 3 2 2 4 4 1 > df[,1] <-unclass(df[,1])</pre>	<p>Что такое data.frame см. в разделе «Фреймы».</p>

<pre>> df[,2] <- unclass(df[,2]) > df a b 1 2 3 2 1 3 3 2 2 4 3 1</pre>	<p>Значения факторов – это «натуральные числа». Обычные числа они «перекодируют» сохраняя порядок. Но при этом</p> <pre>1 -> 1 2 -> 2 4 -> 3</pre>
<pre>> x <- factor(levels= c(10,13,20)) > x[1] <- 10 > x[2] <- 10 > x[3] <- 20 > x[4] <- 25 Предупреждение In `[<-factor`(`*tmp*`, 4, value = 25) : invalid factor level, NAs generated > x # что получилось [1] 10 10 20 <NA> Levels: 10 13 20</pre>	<p>Можно заранее определить, какие уровни будут (названия классов эквивалентностей).</p>
<pre>> a <- factor(c('a', 'b', 'a', 'a', 'b', 'c', 'd')) > b <- a[1:5] > b [1] a b a a b Levels: a b c d > tapply(c(1,2,10,1,5), b, mean) a b c d 4.0 3.5 NA NA</pre>	<p>Тонкость при работе с факторами: если взять подвектор, даже если в нём нет каких-то значений, он всё равно помнит исходные уровни.</p> <p>Вот где это может быть существенно: «лишние NA».</p> <p>Чтобы избавиться от лишних уровней, следовало написать</p> <pre>a[1:5, drop=TRUE].</pre>
<pre>> gl(2,3,7,c('a', 'b')) [1] a a a b b b a Levels: a b > a = gl(2,3,7,c('2', '5')) > as.numeric(levels(a)[a]) [1] 2 2 2 5 5 5 2</pre>	<p>Создание факторов «по шаблону» (число уровней, число повторов, длина).</p> <p>Преобразование факторов в числа.</p>
<pre>> a = gl(2,3,7,c('a', 'b')) > b = gl(3,2,7,c('a', 'b', 'c')) > a [1] a a a b b b a Levels: a b > b [1] a a b b c c a Levels: a b c > interaction(a,b) [1] a.a a.a a.b b.b b.c b.c a.a Levels: a.a b.a a.b b.b a.c b.c</pre>	<p>Объединение факторов.</p>

<pre>> cut(c(1,2,3,4,5,6,7,8,1,2,3), breaks=c(1,5,7)) [1] <NA> (1,5] (1,5] (1,5] (1,5] (5,7] (5,7] <NA> <NA> (1,5] (1,5] Levels: (1,5] (5,7]</pre>	<p>«Разнесение по уровням».</p>
<pre>> x <- c(2,2,5,3,6,5,NA) > xf <- factor(x, levels=2:6) > xf [1] 2 2 5 3 6 5 <NA> Levels: 2 3 4 5 6 > model.matrix(~xf-1) xf2 xf3 xf4 xf5 xf6 1 1 0 0 0 0 2 1 0 0 0 0 3 0 0 0 1 0 4 0 1 0 0 0 5 0 0 0 0 1 6 0 0 0 1 0 attr(,"assign") [1] 1 1 1 1 1 attr(,"contrasts") attr(,"contrasts")\$xf [1] "contr.treatment"</pre>	<p>Построение матрицы характеристических векторов классов эквивалентности.</p>
<pre>> ages = c(12,18,22,20,16,18) > grps = c(1, 2, 1, 1, 2, 3) > tapply(ages, grps, mean) 1 2 3 18 17 18 # двумерные факторы > grps = c(1, 2, 1, 1, 2, 3) > grps2= c(1, 1, 1, 2, 2, 2) > tapply(ages, list(grps,grps2), mean) 1 2 1 17 20 2 18 16 3 NA 18 # если есть NA > grps = c(1, 2, 1, 1, 2, NA) > tapply(ages, grps, mean) 1 2 18 17</pre>	<p>Функция tapply действует как будто второй аргумент факторный.</p>
<pre># продолжение предыдущего # примера > split(ages, list(grps,grps2)) \$`1.1` [1] 12 22</pre>	<p>Функция split похожа на tapply. Она разбивает на группы, но потом никакой функции к ним не применяет, т.е. это первый шаг функции tapply.</p>

<pre>\$`2.1` [1] 18 \$`3.1` numeric(0) \$`1.2` [1] 20 \$`2.2` [1] 16 \$`3.2` [1] 18</pre>	
<pre>> df <- data.frame(sex= c('m','f','m','m','m'), range=c(1,1,1,0,0), age=c(20,25,30,20,20)) > df sex range age 1 m 1 20 2 f 1 25 3 m 1 30 4 m 0 20 5 m 0 20 > split(df\$age, list(df\$sex, df\$range)) \$f.0 numeric(0) \$m.0 [1] 20 20 \$f.1 [1] 25 \$m.1 [1] 20 30 > split(1:6, c(1,2,2,1,1,1)) \$`1` [1] 1 4 5 6 \$`2` [1] 2 3</pre>	<p>Ещё примеры с функцией split...</p> <p>Ещё один пример использования этой функции. Часто применяется в параллельном программировании.</p>

Фреймы

<pre>y = matrix(c(2,3,7,8), nrow=2) colnames(y) = c("Level1", "Level2") rownames(y) = c("Иванов",</pre>	<p>Фреймы. Идеально подходят для хранения матриц «объект-признак». По сути, это таблица, в которой для каждого столбца определён свой тип. Иногда удобно считать,</p>
---	---

<pre> "Петров") year = c(1979, 1981); status = c(TRUE, FALSE); f = data.frame(y, year, status) f Level1 Level2 year status Иванов 2 7 1979 TRUE Петров 3 8 1981 FALSE > f["Иванов", 3] [1] 1979 > f["Иванов",][3] year Иванов 1979 > f["Иванов",][[3]] [1] 1979 > f["Иванов", "year"] [1] 1979 > f\$year [1] 1979 1981 > f[,2] [1] 7 8 > f[,2, drop=FALSE] Level2 Иванов 7 Петров 8 > f[3,] <- list(4, 9, 1985, TRUE) > f Level1 Level2 year status Иванов 2 7 1979 TRUE Петров 3 8 1981 FALSE 3 4 9 1985 TRUE > f[f\$Level2>7,] Level1 Level2 year status Петров 3 8 1981 FALSE 3 4 9 1985 TRUE > subset(f, Level2>7) Level1 Level2 year status Петров 3 8 1981 FALSE 3 4 9 1985 TRUE > subset(f, Level2>7, select=c('year', 'status')) year status Петров 1981 FALSE 3 1985 TRUE </pre>	<p>что это набор списков одинаковой длины. Работа с фреймами очень похожа на работу с матрицами.</p> <p>Обращение к элементам фрейма должно быть понятно после изучения списков.</p> <p>При использовании drop=FALSE результат остаётся фреймом (не конвертируется в вектор).</p> <p>Строки добавляются «как списки».</p> <p>Операции похожи на матричные.</p> <p>Последнюю операцию можно сделать и так...</p> <p>Ещё одно применение функции subset.</p>
<pre> > f Level1 Level2 year status </pre>	<p>Полезная функция – «пометить» строки, которые не содержат неопределённых</p>

<pre>Иванов 2 7 1979 TRUE Петров 3 8 1981 FALSE Сидоров 4 9 1985 NA > complete.cases(f) [1] TRUE TRUE FALSE</pre>	элементов.
<pre>> f2 <- cbind(f, c('m', 'm')) > names(f2)[5] <- 'sex' > f2 Level1 Level2 year status sex Иванов 2 7 1979 TRUE m Петров 3 8 1981 FALSE m</pre>	<p>Конкатенацию, как и для матриц, можно проводить с помощью cbind и rbind.</p> <p>Кстати, здесь добавляется факторный столбец! Сделайте str(f2).</p>
<pre>> attach(f) The following object(s) are masked _by_ '.GlobalEnv': status, year > Level1 [1] 2 3 4 > detach(f) > Level1 Ошибка: объект 'Level1' не найден</pre>	<p>В примере выше можно упростить доступ к столбцам – упоминать только их имя...</p> <p>Но присваивание Level1 = 2 здесь не изменит сам фрейм!</p> <p>Теперь такой доступ опять запрещён.</p>
<pre>> df <- data.frame(row.names = c('Alex', 'Serg'), Exam1 = c(3.4, 4.5), Exam2 = c(4.3, 4.6)) > df Exam1 Exam2 Alex 3.4 4.3 Serg 4.5 4.6 > df[,3] <- df\$Exam1 - df\$Exam2 > df Exam1 Exam2 V3 Alex 3.4 4.3 -0.9 Serg 4.5 4.6 -0.1 > df\$diffEx <- df\$Exam1 - df\$Exam2 > df Exam1 Exam2 V3 diffEx Alex 3.4 4.3 -0.9 -0.9 Serg 4.5 4.6 -0.1 -0.1</pre>	Способы добавления нового столбца.
<pre>> names = c("Alex", "Serg") > age = c(20, 32) > f <- data.frame(names, age) > f names age 1 Alex 20 2 Serg 32 > f\$names [1] Alex Serg Levels: Alex Serg</pre>	Ещё пример фрейма.

<pre>> f <- data.frame(names, age, stringsAsFactors=FALSE) > f names age 1 Alex 20 2 Serg 32 > f\$names [1] "Alex" "Serg" > dim(f) [1] 2 2 > names(f) [1] "names" "age" > row.names(f) [1] "1" "2"</pre>	<p>Если не надо переделывать строки в факторы, а это происходит по умолчанию, то можно воспользоваться опцией stringsAsFactors=TRUE.</p>
<pre>> df <- read.table("C:\\table.txt", sep= ',', header=TRUE) > df A B desc 1 1 2 Иван 2 1 3 Пётр</pre>	<p>Такое чтение из файла переводит данные во фрейм.</p> <p>Содержание текстового файла: A, B, desc 1, 2, Иван 1, 3, Пётр</p>
<pre>> (df1 <- data.frame(x=c(1,1,2,2), y=c('a','b','a','b'))) x y 1 1 a 2 1 b 3 2 a 4 2 b > (df2 <- data.frame(z=c(1,2,3,4), x=c(1,1,2,2), y=c('a','b','a','b'))) z x y 1 1 1 a 2 2 1 b 3 3 2 a 4 4 2 b > merge(df1, df2) x y z 1 1 a 1 2 1 b 2 3 2 a 3 4 2 b 4 > (df3 <- merge(df1, df2, by="x")) x y.x z y.y 1 1 a 1 a 2 1 a 2 b 3 1 b 1 a 4 1 b 2 b</pre>	<p>Для слияния двух фреймов используется функция merge (аналог SQL-вского join).</p>

<pre> 5 2 a 3 a 6 2 a 4 b 7 2 b 3 a 8 2 b 4 b > sapply(df3, sort) x y.x z y.y [1,] 1 1 1 1 [2,] 1 1 1 1 [3,] 1 1 2 1 [4,] 1 1 2 1 [5,] 2 2 3 2 [6,] 2 2 3 2 [7,] 2 2 4 2 [8,] 2 2 4 2 </pre>	<p>К фреймам можно применять функции sapply и lapply, но они действуют на списки-столбцы (в результате часто «фрейм» теряет смысл, поскольку значения столбцов перестают соответствовать друг другу).</p> <p>Здесь показан пример сортировки, в факторах значение меняется на номер класса эквивалентности.</p>
<pre> > (d = data.frame(a=1:3, b=4:6, c=7:9)) a b c 1 1 4 7 2 2 5 8 3 3 6 9 > (t = table(d)) , , c = 7 b a 4 5 6 1 1 0 0 2 0 0 0 3 0 0 0 , , c = 8 b a 4 5 6 1 0 0 0 2 0 1 0 3 0 0 0 , , c = 9 b a 4 5 6 1 0 0 0 2 0 0 0 3 0 0 1 > ftable(t) c 7 8 9 a b 1 4 1 0 0 5 0 0 0 6 0 0 0 </pre>	<p>Обратите внимание на действие функции table. И на вывод с помощью ftable.</p>

<pre> 2 4 0 0 0 5 0 1 0 6 0 0 0 3 4 0 0 0 5 0 0 0 6 0 0 1 > ftable(d) c 7 8 9 a b 1 4 1 0 0 5 0 0 0 6 0 0 0 2 4 0 0 0 5 0 1 0 6 0 0 0 3 4 0 0 0 5 0 0 0 6 0 0 1 </pre>	
<pre> > A <- data.frame(city = c('Moscow', 'Korolev', 'Pushkino') , income=c(9000, 1100, 750)) > B <- data.frame(city = c('Moscow', 'Pushkino', 'Fryzino') , code=c(095, 030, 044)) > merge(A,B,by=1) city income code 1 Moscow 9000 95 2 Pushkino 750 30 > C <- A > m <- match(A\$city, B\$city, 0) > C\$code <- ifelse(m==0, 0, B\$code[m]) > C city income code 1 Moscow 9000 95 2 Korolev 1100 0 3 Pushkino 750 95 </pre>	<p>Два разных объединения фреймов.</p> <pre> city income 1 Moscow 9000 2 Korolev 1100 3 Pushkino 750 </pre> <pre> city code 1 Moscow 95 2 Pushkino 30 3 Fryzino 44 </pre>
<pre> > df = data.frame(x=1:3, y=LETTERS[1:3]) > (df = data.frame(x=1:3, y=LETTERS[1:3])) x y 1 1 A 2 2 B 3 3 C > split(df\$x, df\$y) \$A [1] 1 \$B [1] 2 </pre>	<p>«Разделение» – превращение в список.</p>

<pre>\$C [1] 3</pre>	
<pre>> df name Ex1 Ex2 1 Alex 4 3 2 Serg 4 4 3 Alex 3 5 4 Serg 5 4 5 John 5 4 > aggregate(df[, -1], list(df\$name), mean) Group.1 Ex1 Ex2 1 Alex 3.5 4 2 John 5.0 4 3 Serg 4.5 4</pre>	«Усреднение» информации – функция aggregate .
<pre>> df <- data.frame(names=c("Alex", "Serg", "John"), age=c(14, 16, 13)) > df[order(df\$age),] names age 3 John 13 1 Alex 14 2 Serg 16</pre>	Сортировка по значениям одного из столбцов.

Загрузка данных, запись в файлы [не очень хорошо структурировано]

<pre>> a<-read.table("data1.txt", sep=" ", na.strings="*", header=TRUE) > a Price Floor Area Rooms Age Cent.heat 01 52.00 111 830 5 6.2 no 02 54.75 128 710 5 7.5 no 03 57.50 101 1000 5 4.2 no 04 57.50 NA 800 5 4.0 yes</pre>	Загрузка данных. Отмечаем разделитель и неизвестные значения.
<pre>> a = edit(HousePrice)</pre>	Редактирование.
<pre>> df <- read.table('c:\\temp\\txt.txt') > as.matrix(df) V1 V2 V3 [1,] 3 21 13 [2,] 4 5 22</pre>	Чтобы загрузить матрицу, надо её сначала загрузить как data.frame. В файле 3 21 13 4 5 22
<pre>a <- read.xls("mydata.xls", sheet = "Sheet1")</pre>	Загрузка Excel-данных.

<pre>d <- read.csv("txt.csv") d <- read.csv2("txt.csv") d <- read.delim("foo.txt") d <- read.fwf("txt.fwf")</pre>	<p>Загрузка csv-файла.</p> <p>Загрузка csv-файла при разделении точкой с запятой и использовании запятой при записи чисел.</p> <p>Загрузка файла с разделением табуляцией.</p> <p>Загрузка файла с фиксированными полями.</p>
<pre>> x = matrix(1:6, nr=2) > write(x, file='c:\\temp\\1.txt')</pre>	<pre>1 2 3 4 5 6</pre>
<pre>> x = matrix(1:6, nr=2) > write.table(x, file='c:\\temp\\1.txt')</pre>	<pre>"V1" "V2" "V3" "1" 1 3 5 "2" 2 4 6</pre>
<pre>> x = matrix(1:6, nr=2) > write.table(x, file='c:\\temp\\1.txt', row.names=F, col.names=F)</pre>	<pre>1 3 5 2 4 6</pre>
<pre>> x = matrix(1:6, nr=2) > write.csv(x, file='c:\\temp\\1.txt')</pre>	<pre>"", "V1", "V2", "V3" "1", 1, 3, 5 "2", 2, 4, 6</pre>
<pre>> x = matrix(1:6, nr=2) > write.csv2(x, file='c:\\temp\\1.txt')</pre>	<pre>""; "V1"; "V2"; "V3" "1"; 1; 3; 5 "2"; 2; 4; 6</pre>
<pre>> save(x, file = 'c:\\temp\\1.txt') > rm(x) > x Ошибка: объект "x" не найден > y <- load(file = 'c:\\temp\\1.txt') > x [,1] [,2] [,3] [1,] 1 3 5 [2,] 2 4 6 > y [1] "x"</pre>	<p>Внутренний формат R – не текстовый файл (можно сохранять несколько переменных)</p> <p>Надо просто: <code>load(file = 'c:\\temp\\1.txt')</code></p> <p>Загрузка происходит в переменную x!</p>
<pre>> df name age 1 John 15 2 Alex 17 > write.table(df, 'txt.txt') write.table(df, 'txt.txt', quote=FALSE, col.names=FALSE, row.names=FALSE, sep=';')</pre>	<p>Первый способ записи в файл. В файле:</p> <pre>"name" "age" "1" "John" 15 "2" "Alex" 17</pre> <p>Кстати, <code>quote=FALSE</code> убирает кавычки в строках. В файле:</p> <pre>John;15 Alex;17</pre>
<pre>> cat(file='txt.txt', c(1,4,7), "\nstr\n") > cat(file='txt.txt', 1:5,</pre>	<p>Другие способы записи в файл. В файле:</p> <pre>1 4 7 str</pre>

<pre>append=TRUE) fl <- file("txt.txt", "w") writeLines(c("str1", "str2"), fl) close(fl)</pre>	<p>1 2 3 4 5</p> <p>В файле:</p> <pre>str1 str2</pre>
<pre>read.table("filename.txt", colClasses = c("numeric", rep("character", 10))) read.table("foo.csv", header = TRUE, sep = ",", na.strings = c("#N/A!", "NA", "@NA"), quote = '"', comment.char = "")</pre>	<p>При загрузке больших файлов лучше сразу указать типы столбцов.</p>
<pre>> scan('c:\\temp\\txt.txt') Read 5 items [1] 1 2 3 4 5 > scan('c:\\temp\\txt.txt', sep='\n') Read 2 items [1] 12 345 > scan('c:\\temp\\txt.txt', what='') Read 5 items [1] "1" "2" "3" "4" "5" > scan('c:\\temp\\txt.txt', what='', sep='\n') Read 2 items [1] "1 2" "3 4 5"</pre>	<p>Различные способы считывания файлов. На практике надо делать так</p> <pre>m <- scan(...)</pre> <p>второй аргумент – «тип» считываемых данных, sep – разделитель.</p> <p>В файле</p> <pre>1 2 3 4 5</pre>
<pre>> a <- scan() 1: 12 2: 2 3 4: Read 3 items > a [1] 12 2 3 > s <- readline("Ваш пароль:") Ваш пароль:mmр > s [1] "mmр"</pre>	<p>Ввод данных с клавиатуры выполняется той же функцией.</p> <p>Можно сделать ввод одной строки.</p>
<pre>> print("строка\nstr") [1] "строка\nstr" > cat("строка\nstr") строка str > cat("строка", "\n", "str", sep='+') строка+ +str</pre>	<p>Различные способы вывода на экран.</p> <p>Функция cat правильно интерпретирует символы вида \n, кроме того, конкатенирует все свои аргументы.</p>
<pre>> txt <- scan('temp1.txt', "")</pre>	<p>Функция scan позволяет считать файл «по</p>

<pre>Read 8 items > txt [1] "\$1." "Знакомство" "с" "R" [5] "\$2." "Работа" "с" "системой" > words <- split(1:length(txt), txt) > words \$`\$1.` [1] 1 \$`\$2.` [1] 5 \$R [1] 4 \$Знакомство [1] 2 \$Работа [1] 6 \$c [1] 3 7 \$системой [1] 8</pre>	<p>Словам».</p> <p>Преобразование слов в список.</p>
<pre>> adr <- "http://archive.com/db/" > adr <- paste(uci, "echo/echo.data", sep="") > ecc <- read.csv(adr)</pre>	<p>Чтение данных «из Интернета».</p>
<pre>> search() [1] ".GlobalEnv" "package:stats" "package:graphics" [4] "package:grDevices" "package:utils" "package:datasets" [7] "package:methods" "Autoloads" "package:base"</pre>	<p>Текущие загруженные пакеты.</p>
<pre>> library()</pre>	<p>Список доступных пакетов (содержатся в директории, но, возможно, не загружены).</p>
<pre># создать файл > cat("NAME", "1:John", "2:Paul", file = "foo", sep = "\n") # считать число полей > count.fields("foo", sep = ":")</pre>	<p>Полезная функция count.fields – считает число полей в каждой строке файла. Поля разделены символом sep. Часто имеет смысл такая конструкция: table(count.fields("foo", sep = ":")).</p>

<pre>[1] 1 2 2 > unlink("foo") # удалить</pre>	
<pre>D <- read.csv("data.csv", header=TRUE, stringsAsFactors=FALSE) print(unique(sort(D[,col])))</pre>	<p>После загрузки файла полезно взглянуть на уникальные значения в каждом столбце...</p>
<pre>> fl <- file("c:\\temp\\txt.txt", "r") > readLines(fl,2) # 2 строки [1] "Alex, 2" "Serg, 3" > seek(con=fl, where=2) [1] 9 > readLines(fl, 1) [1] "ex, 2" while(TRUE) { rl <- readLines(fl,n=1) if (length(rl) == 0) break else print(rl) } [1] "Serg, 3" [1] "John, 12" > close(fl)</pre>	<p>Показано чтение файла по строкам. С помощью seek можно перемещаться по файлу.</p> <p>Сначала создаётся соединение, подробности можно узнать, используя команду ?connection</p> <p>Например, можно создать срединение командой url().</p> <p>После работы с файлом (соединением) надо «его закрыть».</p>
<pre>> setwd('c:\\temp\\') > getwd() [1] "c:/temp" > file.info('txt.txt') size isdir mode mtime ctime txt.txt 12 FALSE 666 2013-05- 03 16:47:27 2013-05-03 15:05:41 atime exe txt.txt 2013-05-03 15:05:41 no > file.exists('txt.txt') [1] TRUE > dir() [1] "1363498.key" [2] "16.Chetviorkin.pdf" [3] "txt.txt"</pre>	<p>Работа с файлами и директорией. Задать рабочую директорию, узнать рабочую директорию, информация о файле,</p> <p>существует ли файл, файлы директории.</p>
<pre>sumtree <- function(drtr) { tot <- 0 # все имена файлов (+подкаталоги) fls <- dir(drtr,recursive=TRUE) for (f in fls) {</pre>	<p>Пример из [ART] – найти сумму данных во всех файлах каталога.</p>

<pre># f – директория? f <- file.path(drtr,f) if (!file.info(f)\$isdir) { tot <- tot + sum(scan(f,quiet=TRUE)) } } return(tot) }</pre>	
--	--

Советы по убыстрению кода

<pre>> a = runif(1000000) > b = runif(1000000) > system.time(a <- a*b-b^0.7) пользователь система прошло 0.13 0.03 0.16 > system.time(for (i in 1:1000000) a[i] <- a[i]*b[i]- b[i]^0.7) пользователь система прошло 5.63 0.00 5.63 > library(compiler) > f <- function() for (i in 1:1000000) a[i] <- a[i]*b[i]- b[i]^0.7 > cf <- cmpfun(f) > system.time(cf()) пользователь система прошло 1.11 0.00 1.11</pre>	<p>Всегда пользуйтесь встроенными средствами параллелизации.</p> <p>Здесь происходят обращения к различным функциям: for, «:» (двоеточие это тоже функция), «<-».</p> <p>Можно использовать «byte code compiler», чтобы ускорить код.</p>
<pre>> a <- 2 > b <- a > tracemem(a) [1] "<0x0c9d7ce0>" > tracemem(b) [1] "<0x0c9d7ce0>" > b <- 3 > tracemem(a) [1] "<0x0c9d7ce0>" > tracemem(b) [1] "<0x0c347d28>"</pre>	<p>После присваивания новая переменная не создалась (в том смысле, что ссылки a и b ссылаются на один участок памяти). Но когда переменная b изменилась – под неё выделилось новое место.</p>
<pre>Rprof() # вызов функции Rprof(NULL) summaryRprof()</pre>	<p>Для того, чтобы узнать, какие функции тормозят программу, используйте профайлинг.</p>
	<p>RMySQL (доступ к БД SQL), biglm (регрессия на больших данных), ff (может хранить данные на диске), bigmemory (может хранить данные на диске и в основной памяти).</p>

Функция by!!!

Полезные функции

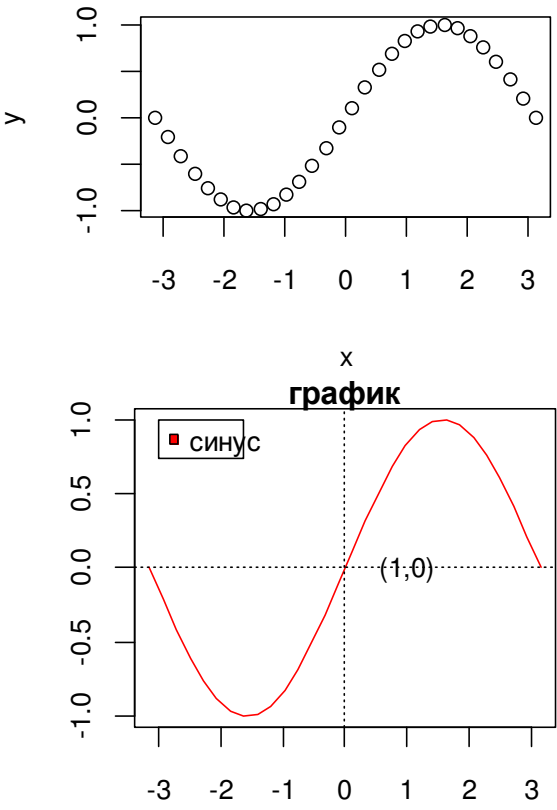
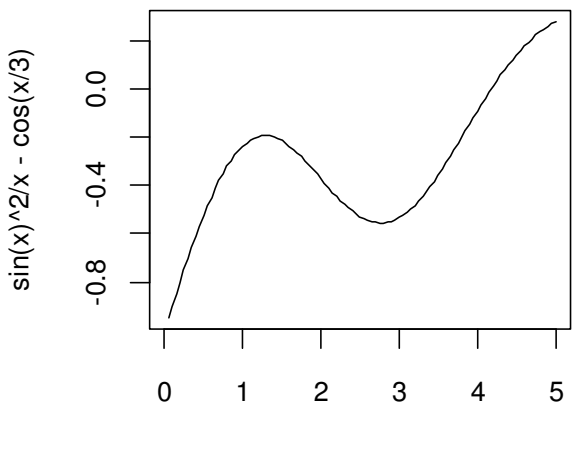
<pre>> tf = c(TRUE, FALSE) > tt = c(TRUE, TRUE) > ff = c(FALSE, FALSE) > c(any(tf), all(tf)) [1] TRUE FALSE > c(any(tt), all(tt)) [1] TRUE TRUE > c(any(ff), all(ff)) [1] FALSE FALSE</pre>	<p>Логические функции: any – хотя бы один аргумент TRUE, all – все аргументы TRUE.</p>
<pre>> a = c(1,2,2,1,3,2,1,2,3) > b = c(1,1,2,2,1,2,1,1,2) > table(a,b) b a 1 2 1 1 2 1 2 2 2 2 3 3 1 1 > table(a) a 1 2 3 3 4 2 > table(b) b 1 2 5 4 > addmargins(table(a,b), FUN=c('max','min')) Margins computed over dimensions in the following order: 1: a 2: b b a 1 2 min 1 1 2 1 1 2 2 2 2 2 3 3 1 1 1 max 2 2 2</pre>	<p>Вывод статистики числа вхождений. – для пар – для вхождений элементов Применение функций «по размерностям».</p>
<pre>> findInterval(22, c(1,10,20,30,40)) [1] 3</pre>	<p>Найти номер интервала, в котором находится значение.</p>
<pre>> do.call(sum, list(1,2,3)) [1] 6</pre>	<p>Если заранее не известно число аргументов функции, то можно для её вызова использовать do.call.</p>
<pre>s1 <- 11 s2 <- 20 s3 <- -9 m = c() for (i in 1:3)</pre>	<p>get – очень мощная функция! Она позволяет получать объект по его имени. Нельзя здесь сделать так: get(s) <- i.</p>

<pre>{ s = paste('s',i,sep=' ') m[i] <- get(s) } m [1] 11 20 -9 # ещё одно применение get("%o%")(2,3) [,1] [1,] 6</pre>	
--	--

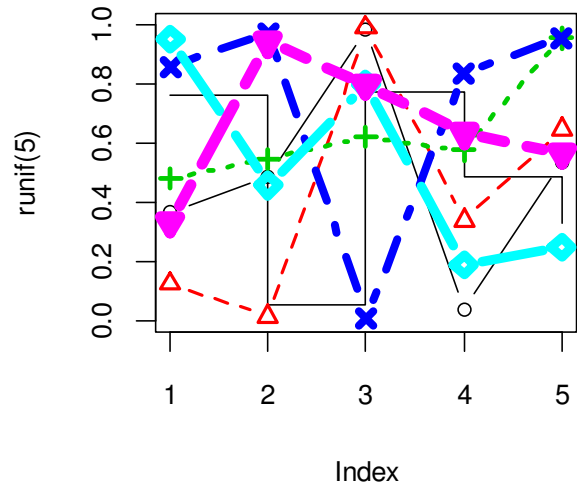
Графика

<pre>dev.new() # новое dev.set(2) # перейти windows dev.cur() # текущее windows 2 dev.list() # все windows windows windows 2 3 4 dev.off(3) # закрыть windows 2 dev.off() # закрыть тек. windows 4 graphics.off() # закрыть всё</pre>	<p>Функции для работы с графическими окнами.</p>
<pre>windows()</pre>	В среде WINDOWS – создать новое графическое окно.
<pre>locator()</pre>	Функция, которая позволяет запоминать координаты кликов по графическому окну.
<pre>> plot(c(1,2,1,2)) > rc = recordPlot() > plot(c(3,2,1,3)) > replayPlot(rc)</pre>	Способ сохранить график, а потом восстановить.

<pre>x = seq(-pi, pi, len = 30) y = sin(x) plot(x, y) plot(y~x) # можно так # цвет, надписи, легенда plot(x, y, type="l", col="red", xlab="time", ylab="", main="график") text(1,0,"(1,0)") legend(-3,1,"синус","red") # добавить сетку -</pre>	<p>Обычный график функции. Обратите внимание на «чисто R-овскую» запись y~x (изобразить y как функцию от x).</p>
---	---

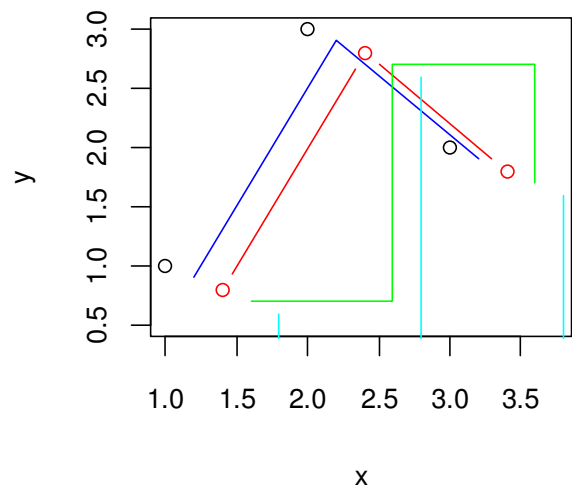
<pre># 2 линии abline(v=0, lty=3) abline(h=0, lty=3)</pre>	 <p>Попробуйте выполнить команду</p> <pre>lines(x, -y, type="l", col="blue")</pre>
<pre>curve(sin(x)^2/x-cos(x/3), 0, 5)</pre>	<p>Упрощённый вывод нужного графика.</p> 
<pre>plot(runif(5), ylim=c(0,1), type='s', col='black') for (i in 1:6) { lines(runif(5), col=i, # цвет lwd=i, # толщина lty=i, # тип линий pch=i, # тип точек type='b') # вывод линий и точек</pre>	<p>Разный цвет, стиль и толщина. Обратите внимание на начальный вызов <code>plot</code>, а затем уже <code>lines</code>. Правильнее в таких случаях первый вызов делать с <code>type='n'</code> (ничего не рисует).</p>

```
}
```



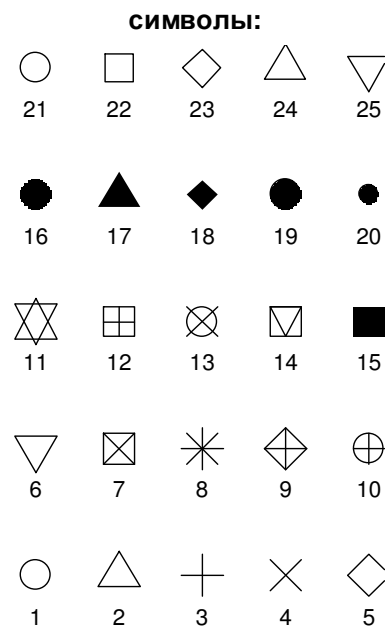
```
x = 1:3
y = c(1,3,2)
plot(x, y, type = 'p',
col='black', xlim=c(1, 3.75),
ylim=c(0.5,3)) # точки
lines(x+0.2, y-0.1, type = 'l',
col='blue') # линии
lines(x+0.4, y-0.2, type = 'b',
col='red') # линии и точки
lines(x+0.6, y-0.3, type = 's',
col='green') # кус-лин
lines(x+0.8, y-0.4, type = 'h',
col='cyan') # гистограмма
lines(x+1, y-0.5, type = 'n') #
не выводить
```

Разные способы вывода графиков.

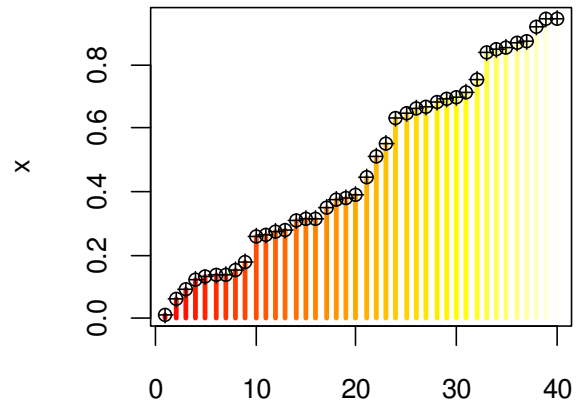


```
op <- par(mar=c(1,0,2,0)+.1)
plot(0,0,
xlim = c(0.85,5),
ylim = c(-.5,4),
axes = FALSE,
xlab = '', ylab = '',
main = "СИМВОЛЫ:")
for (i in 0:4) {
for (j in 1:5) {
n <- 5*i+j
points(j, i,
pch = n,
cex = 3)
text(j,i-0.3, as.character(n))
}
}
par(op)
```

Вывод различных символов для
 использования в графиках.



```
> x = sort(runif(40))
> plot(x, type = 'h', lwd = 3,
col = heat.colors(length(x)),
xlab='')
> lines(x, type='b', pch=10)
```

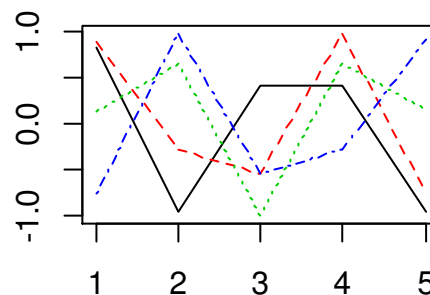


Для вывода различных цветов можно также использовать

```
rainbow
heat.colors
terrain.colors
topo.colors
cm.colors
```

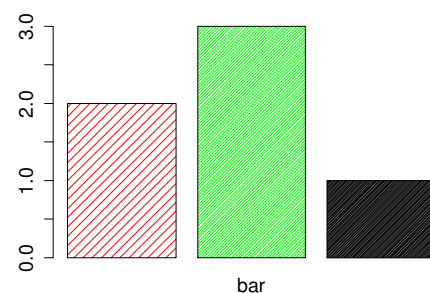
```
> m <- matrix(sin(1:20), nrow=4)
> matplot(1:5, t(m), type='l',
xlab='', ylab='')
```

Обычные графики...



Обратите внимание: все строится одной командой!

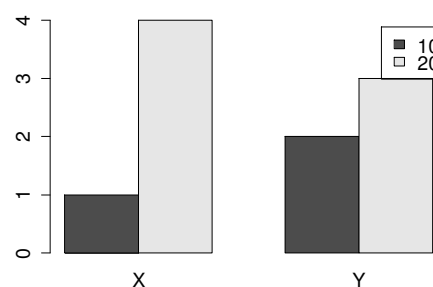
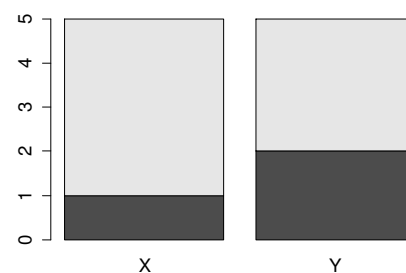
```
> a = c(2, 3, 1)
> names(a) = c("a", "b", "c")
> barplot(a, name="bar",
col=c("red", "green", "black"),
density=c(10, 30, 70))
```




```
a = c(2,3,1)
names(a) = c("a","b","c")
barplot(a, name="bar",
col=c("red","green","black"),
density=c(10,30,70))

> a = c(1,4,2,3)
> a <- matrix(a, nrow=2)
> a
      [,1] [,2]
[1,]    1    2
[2,]    4    3
> colnames(a) = c("X","Y")
> rownames(a) = c(10,20)
> barplot(a)

> barplot(a, beside=TRUE,
legend.text=TRUE)
```

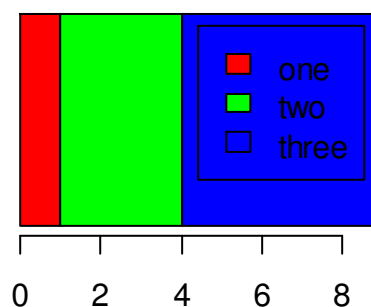


Обратите внимание, что легенда «налезает» на график.

```
> x = c(1,3,5)
> names(x) = c('one', 'two',
'three')
> barplot(as.matrix(x), horiz =
TRUE, col = rainbow(length(x)),
legend.text = TRUE)

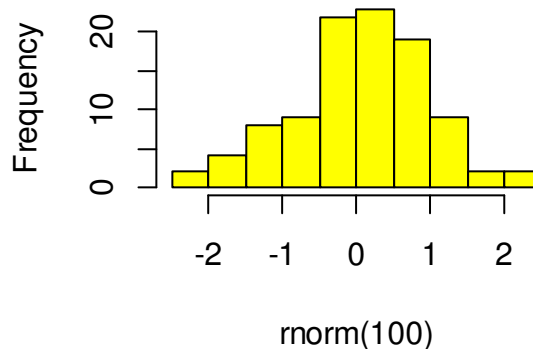
> title(main = "Bar plot")
```

Bar plot



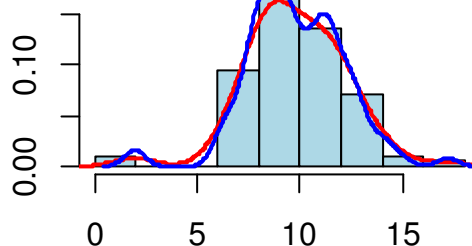
```
> hist(rnorm(100), breaks=7,
col='yellow')
```

Histogram of rnorm(100)



```
> x = rnorm(100, mean=0, sd=1) +
rnorm(100, mean=10, sd=2)
> hist(x, probability=TRUE,
breaks=10, col="light blue",
xlab="", ylab="", main="")
> points(density(x, bw=1),
type='l', lwd=2, col="red")
> points(density(x, bw=.5),
type='l', lwd=2, col="blue")
```

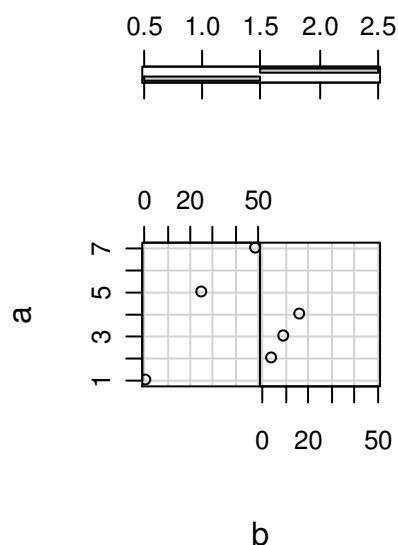
Гистограмма и различные аппроксимации плотности.



```
> a = c(1, 5, 7, 2, 3, 4)
> b = a^2
> coplot(a ~ b | c(1, 1, 1, 2, 2, 2))
```

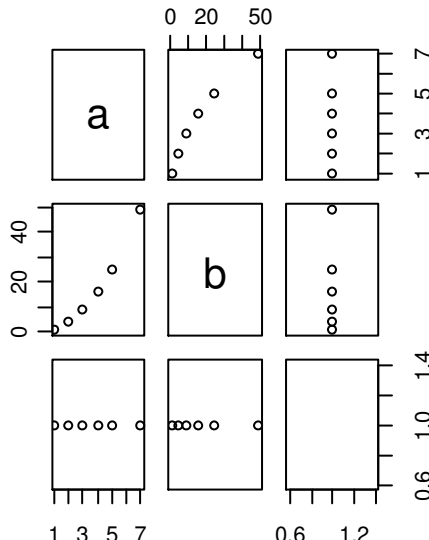
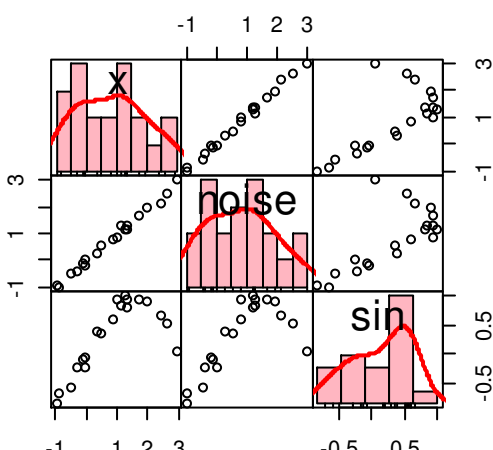
График по уровням третьего вектора.

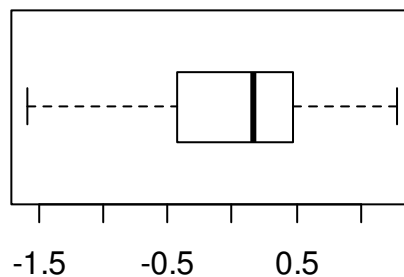
Given : c(1, 1, 1, 2, 2, 2)



```
> m = t(rbind(a, b, 1))
```

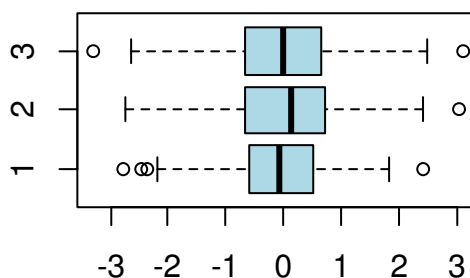
Попарные столбцовые графики.

<pre>> m a b [1,] 1 1 1 [2,] 5 25 1 [3,] 7 49 1 [4,] 2 4 1 [5,] 3 9 1 [6,] 4 16 1 > pairs(m)</pre>	
<pre># ГОТОВИМ ДАННЫЕ > x <- runif(20, min=-1, max=3) > x <- cbind(x, x+rnorm(20)/10, sin(x)+rnorm(20)/10); > colnames(x) <- c('x', 'noise', 'sin') > pairs(x, gap=0, # функция на диагонали diag.panel = function (x, ...) { par(new = TRUE) hist(x, col = "light pink", probability = TRUE, axes = FALSE, main = "") lines(density(x), col = "red", lwd = 2) rug(x) })</pre>	<p>Небольшое изменение стандартного вывода (взято из [1]).</p> 
<pre>x = rnorm(20) boxplot(x, range=0, horizontal=TRUE)</pre>	<p>«Ящик с усами» (минимум, максимум, медиана, первая и третья квантили).</p>



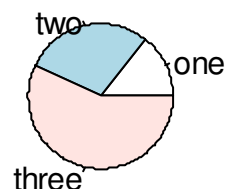
```
> x = c(rnorm(100), rnorm(200),  
rnorm(300))  
> y = factor(c(rep(1, 100),  
rep(2, 200), rep(3, 300)))  
boxplot(x~y, horizontal=TRUE,  
col='light blue')
```

Ящик с усами для трёх подвыборок одной выборки (задаются факторным вектором). Отмечены выбросы.

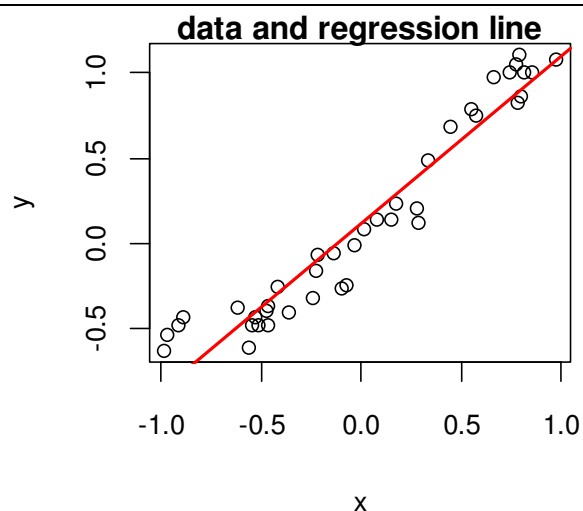


```
x = c(1, 2, 4)  
names(x) <- c('one', 'two',  
'three')  
pie(x)
```

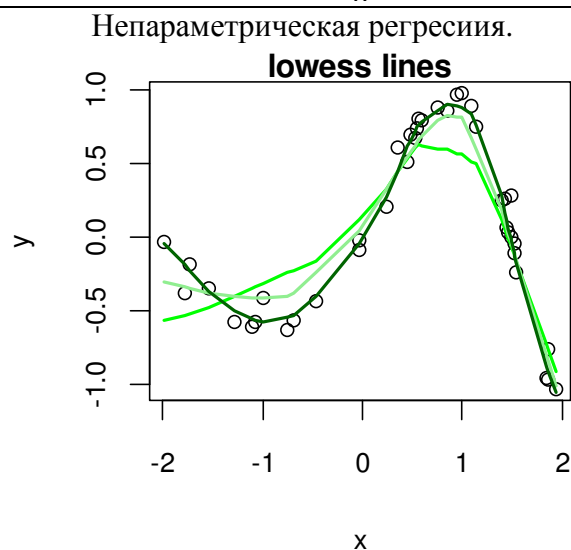
График-пирог.



```
# данные
> x = runif(40, min=-1, max=1)
> y = sin(0.6*x*(x+2)) +
rnorm(40, sd=0.1)
# изобразить
> plot(y ~ x, main = "data and
regression line")
# линия регрессии
> abline(lm(y ~ x), col = 'red',
lwd = 2)
```



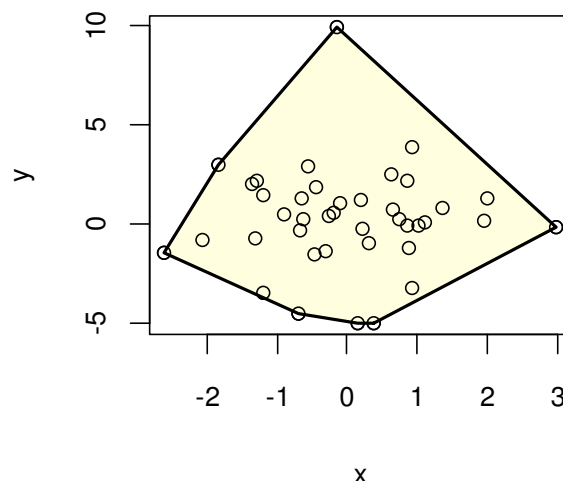
```
> x = runif(40, min=-2, max=2)
> y = sin(0.6*x*(x+2)) +
rnorm(40, sd=0.1)
# изобразить
> plot(y ~ x, main = "lowess
line")
# линия непараметрической
регрессии
> lines(lowess(x,y), col =
'green', lwd = 2)
> lines(lowess(x,y,f=0.5), col =
'lightgreen', lwd = 2)
> lines(lowess(x,y,f=0.2), col =
'dark green', lwd = 2)
```



См. также функцию **loess**.

```
> x = rnorm(40)
> y = rnorm(40, sd=3)
> plot(y~x)
> T = cbind(x, y)
> polygon( T[chull(T),],
col="light yellow", lwd=2)
points(x, y)
```

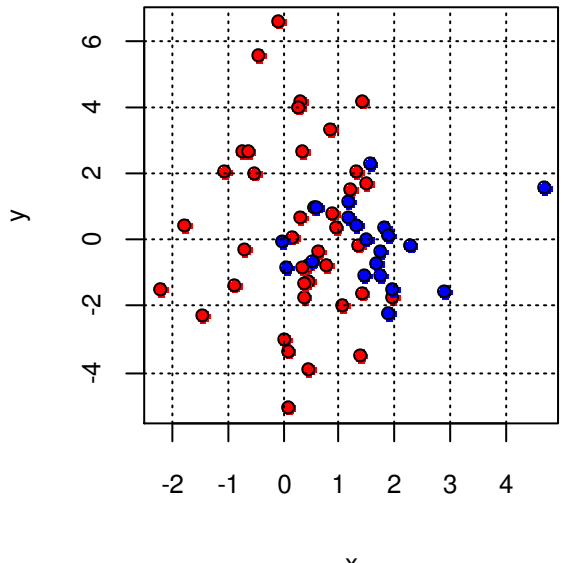
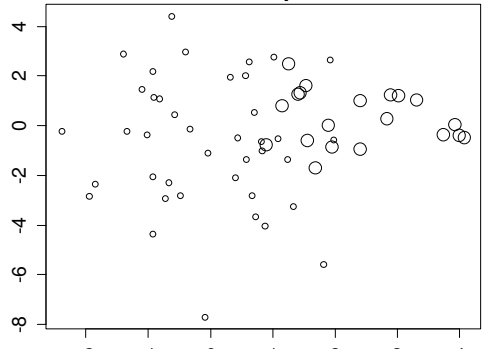
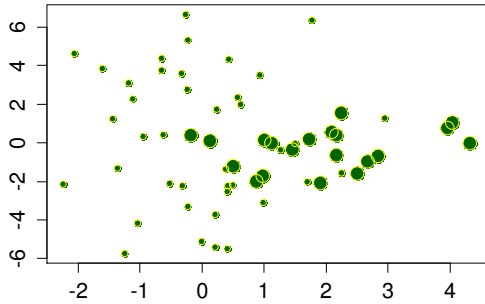
Выпуклая оболочка точек.

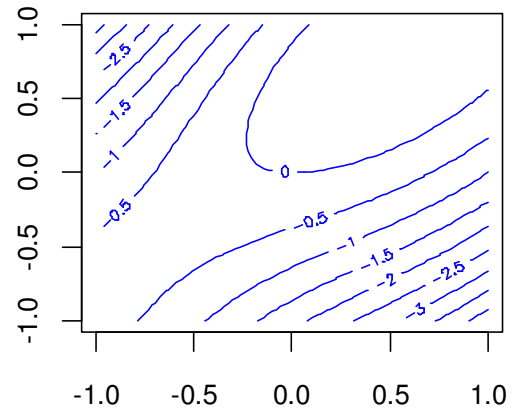


```
op <- par(mar=c(3,3,0,0)+1,
ps=10)
# отступы + шрифт

x = c(rnorm(40), rnorm(20,
mean=2))
y = c(rnorm(40, sd=3),
```

«Классический» вывод для задач классификации.

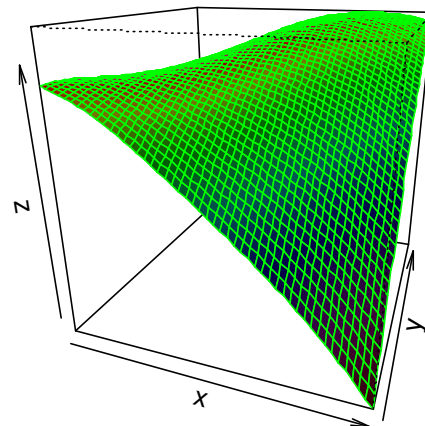
<pre> rnorm(20)) plot(x,y, bg=c('red', 'blue')[c(rep(1,40), rep(2,20))], pch=21) # pch=21 нужно, чтобы выводились закрашенные точки grid(col='black') # сетка par(op) </pre>	 <p>A scatter plot showing two groups of data points. The first group consists of 40 red filled circles, and the second group consists of 20 blue filled circles. The points are scattered across a coordinate system with x-axis from -2 to 4 and y-axis from -4 to 6. A black dashed grid is overlaid on the plot.</p>
<pre> x = c(rnorm(40), rnorm(20, mean=2)) y = c(rnorm(40, sd=3), rnorm(20)) z = c(rep(1,40), rep(2,20)) plot(x, y, cex = z, xlab = "", ylab = "", main = "Bubble plot") </pre>	 <p>A bubble plot titled "Bubble plot". The x-axis ranges from -2 to 4 and the y-axis from -6 to 4. Data points are represented as open circles. The size of each circle is proportional to the value of the variable z, which has two discrete values (1 and 2), resulting in two distinct sizes of circles.</p>
<pre> x = c(rnorm(40), rnorm(20, mean=2)) y = c(rnorm(40, sd=3), rnorm(20)) z = c(rep(1,40), rep(2,20)) plot(x, y, cex = z, xlab = "", ylab = "", col='dark green', pch=16) points(x, y, cex = z, col='yellow') </pre>	 <p>A scatter plot with two groups of data points. The first group consists of 40 dark green filled circles, and the second group consists of 20 yellow filled circles. The points are scattered across a coordinate system with x-axis from -2 to 4 and y-axis from -6 to 6.</p>
<pre> N <- 50 x <- seq(-1, 1, length=N) y <- seq(-1, 1, length=N) xx <- matrix(x, nr=N, nc=N) yy <- matrix(y, nr=N, nc=N, byrow=TRUE) z <- exp(-xx^2)*sin(yy)-(yy- xx)^2 contour(x, y, z, col='blue') </pre>	<p>Линии уровня функции.</p>



```
# устанавливает ширину границ
op <- par(mar=c(0,0,0,0)+1)

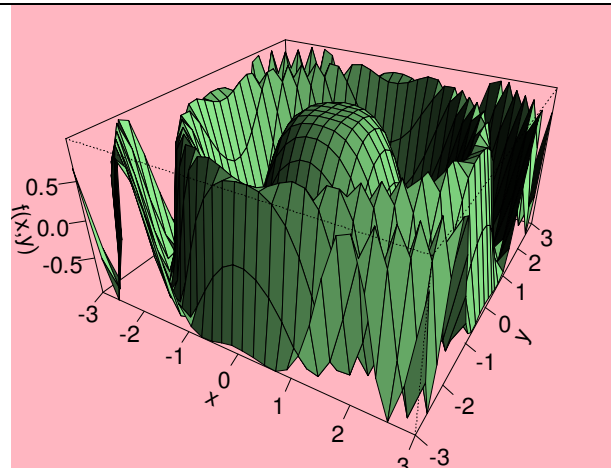
persp(x, y, z,
theta = 20, phi = 20,
shade = .5,
col = rainbow(N),
border = "green")
par(op)
```

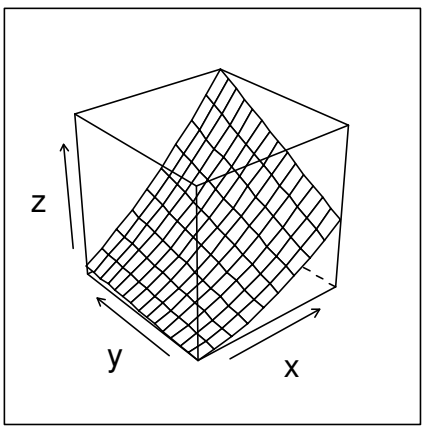
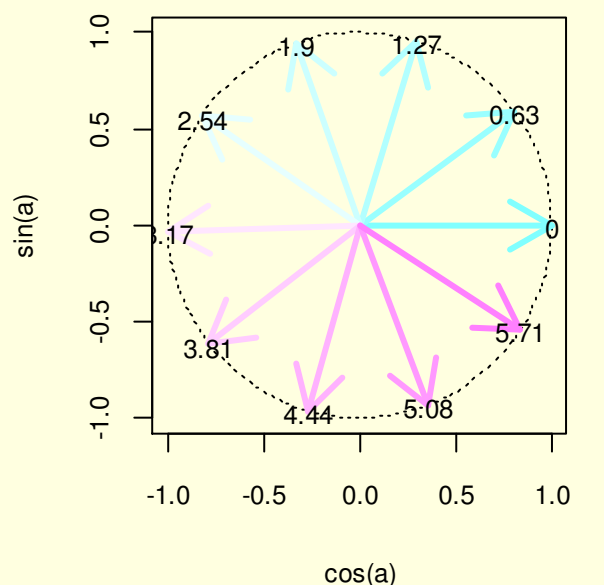
График самой функции.
 [СМ. ПОЛЕЗНЫЙ СПОСОБ УМЕНЬШИТЬ
 ГРАНИЦЫ!!!!]

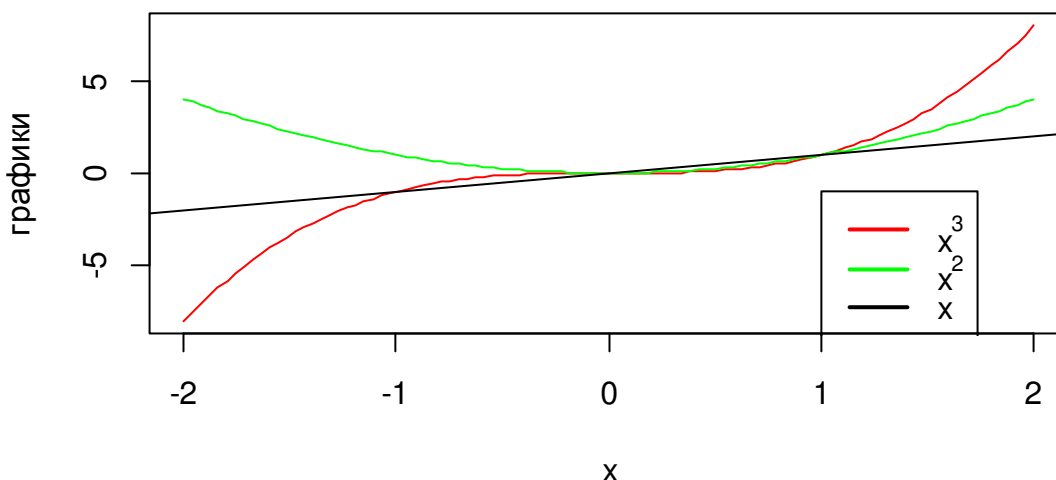


```
# подготовка данных
x <- seq(-3, 3, length= 30)
y <- x
f <- function(x,y)
{cos(x^2+y^2)}
z <- outer(x, y, f)
# это другой способ!

# рисование
op <- par(bg = "light pink",
mar=c(0,0,0,0)+1)
persp(x, y, z,
theta = 30, phi = 30,
expand = 0.5,
col = "light green",
ltheta = 120,
shade = 0.75,
ticktype = "detailed",
xlab = "x", ylab = "y", zlab =
"f(x,y)"
)
```



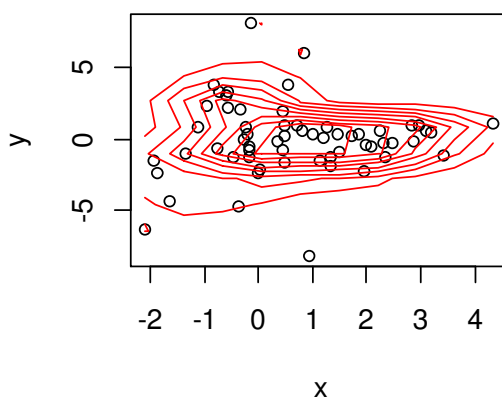
<pre> par(op) library(lattice) a <- 1:10 b <- 1:15 eg <- expand.grid(x=a,y=b) eg\$z <- eg\$x^2 + eg\$y * eg\$y wireframe(z ~ x+y, eg) </pre>	
<pre> # обратите внимание на ps=10 - # ШРИФТ op <- par(bg = "light yellow", mar = c(3,3,0,0)+1, ps=10) a <- seq(0,2*pi,length=100) # чертим окружность plot(cos(a), sin(a), type = 'l', lty = 3) I = seq(from=1, to=length(a), by=10) cols = cm.colors(length(I)) # проводим стрелки for (j in 1:length(I)) { i = I[j] arrows(0,0, cos(a[i]), sin(a[i]), col=cols[j], lw=3) text(cos(a[i]), sin(a[i]), round(a[i],2)) } par(op) </pre>	



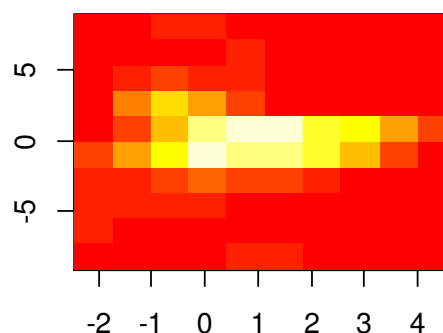
Вывод графиков различных функций. Обратите внимание на `expression` – функция, которая позволяет вставлять математические выражения.

```
x <- seq(from=-2, to=2, length=100) # "сетка точек"
y1 <- x^3 # первая функция
y2 <- x^2 # вторая
plot( y1~x, type = 'l', col = 'red', xlim = c(-2,2),
      ylab="графики")
lines( y2~x, type='l', col='green')
abline(0,1) # линия
# добавить легенду
legend( 1, -1,
        c(expression(x^3), expression(x^2), "x"),
        lwd=2,
        col=c("red", "green", "black")
      )
```

Оценка плотности



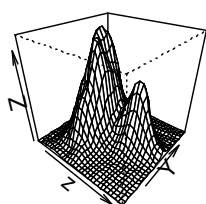
Оценка плотности



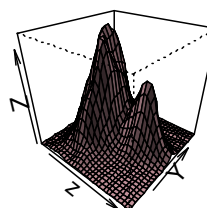
```
library(MASS)
# данные
```

```
x = c(rnorm(40), rnorm(20, mean=2))
y = c(rnorm(40, sd=3), rnorm(20))
T = cbind(x, y)
# оценка плотности
z <- kde2d(x,y, n=10)
# вывод первого изображения
plot(x,y, main = "Оценка плотности")
contour(z, col = "red", drawlabels = FALSE, add = TRUE)
# вывод второго
image(z, main = "Оценка плотности")
```

Оценка плотности

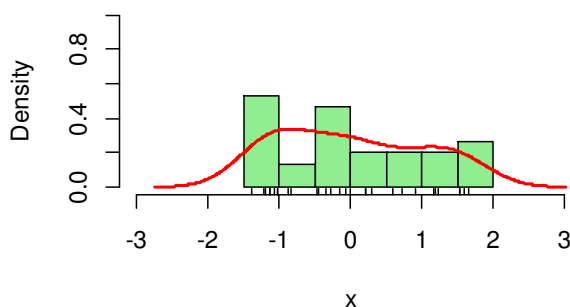


Оценка плотности

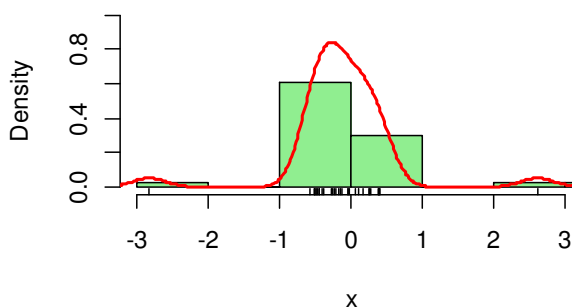


```
# увеличим выборку и выведем график плотности
x = c(rnorm(400), rnorm(100, mean=4))
y = c(rnorm(400, sd=3), rnorm(100))
z <- kde2d(x,y, n=30)
persp(z, main = "Оценка плотности", phi=30, theta=40)
# для вывода второго графика
persp(z, main = "Оценка плотности", phi=30, theta=40, col='pink',
shade=0.4)
```

Равномерное распределение



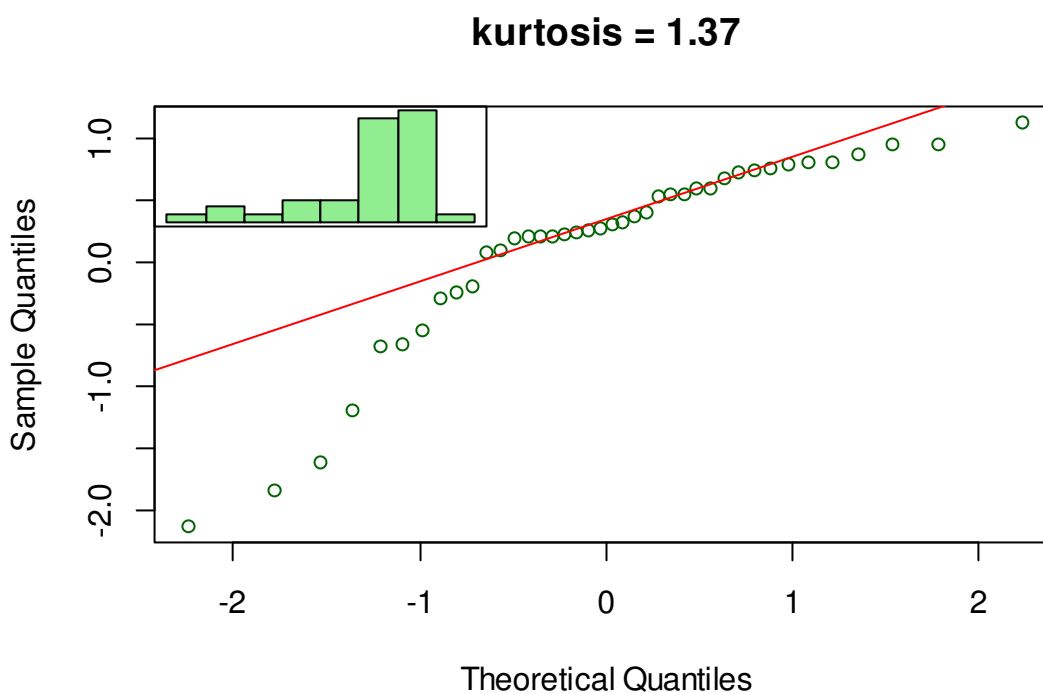
Р.р. с выбросами



Небольшая переделка примера из [1]. Вывод осуществляется сразу на «две половинки» изображения. Выводится гистограмма, аппроксимация плотности и сама выборка.

```
N <- 30 # размер выборки
set.seed(2)
x1 <- runif(N) # равномерное распределение
x2 <- c(x1, -2, 3, 4) # + выбросы
```

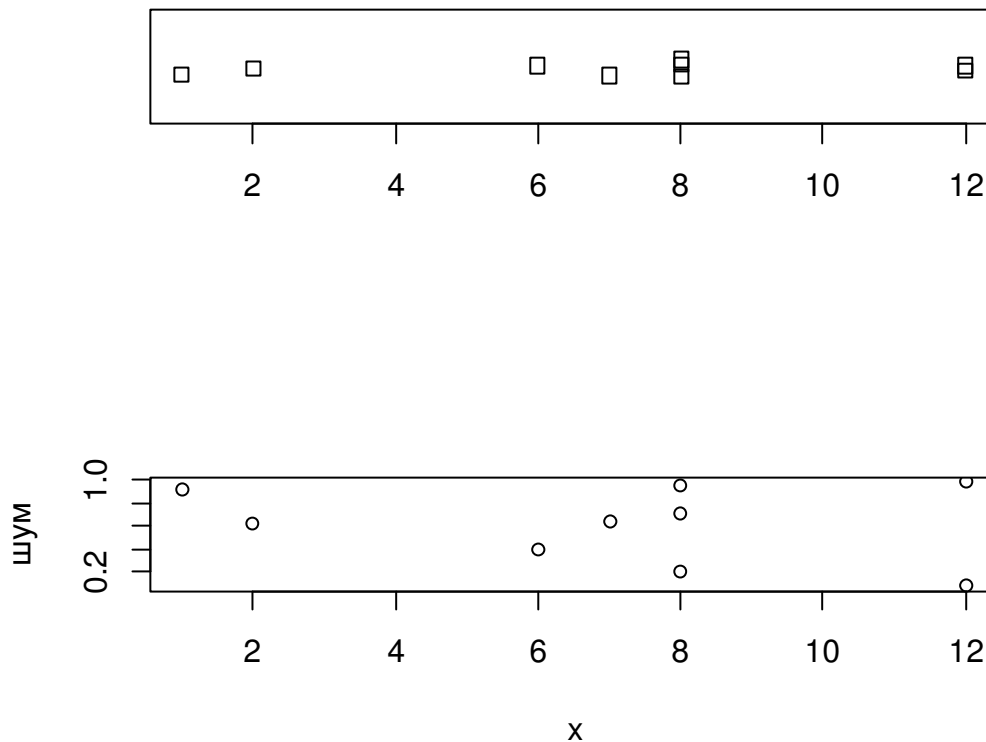
```
f <- function (x, ...)  
{  
  x <- (x - mean(x)) / sd(x) # стандартизация  
  # вывод гистограммы  
  hist(x,  
    col = "light green",  
    xlim = c(-3, 3),  
    ylim = c(0, 1),  
    probability = TRUE,  
    ...  
  )  
  # вывод аппроксимации плотности  
  lines(density(x), col = "red", lwd = 2) # плотность  
  rug(x) # + выборка  
}  
  
op <- par(mfrow=c(1,2))  
f(x1, main = "Равномерное распределение")  
f(x2, main = "Р.р. с выбросами")  
par(op)
```



Небольшая переделка примера из [1]. Оценка «похожести на нормальное распределение». Очень интересно смотрится гистограмма в левом верхнем углу.

```
library(e1071) # подключение библиотеки (там есть ф-я kurtosis)  
n = 40  
x <- c(rnorm(n-20), runif(20))  
qqnorm(x, main=paste("kurtosis =", round(kurtosis(x), digits=2)),  
  col="dark green")  
qqline(x, col="red")
```

```
# теперь выводим в углу рисунка
op <- par(fig=c(0, 0.5, 0.4, 1), new=TRUE)
hist(x, probability=T, col="light green", xlab="", ylab="",
main="", axes=F)
box()
par(op)
```



Вывод одномерной выборки. Шум нужен, чтобы точки «не накладывались» друг на друга.

```
op <- par(mfrow=c(2,1))
x = c(1,2,6,7,8,8,12,12)
stripchart(x, jitter=TRUE, method="jitter")
plot( runif(length(x)) ~ x, ylab="шум")
par(op)
```

Методы главных и независимых компонент

```
# порождение выборки
x = rnorm(100);
y = rnorm(100);
T = cbind(x+2*y, x-3*y);
a = prcomp(T) # метод PCA
# рисование
op <- par(mfrow=c(2,1),
mar=c(0,0,0,0)+2)
plot(T[,1], T[,2], col='dark
red', pch=16, xlim=c(-10,10),
ylim=c(-10,10), xlab='',
```

Демонстрация метода главных компонент
 (первая картинка – исходная выборка, вторая
 – приведение к главным осям).

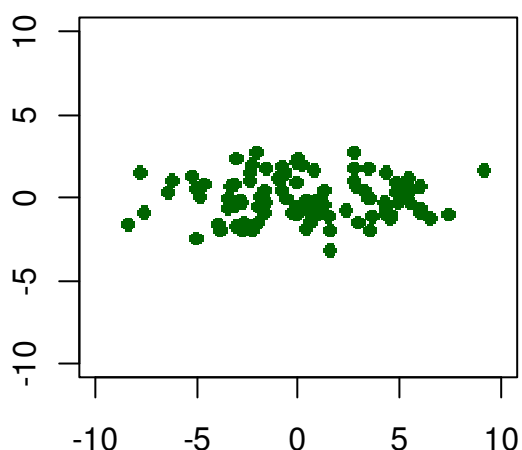
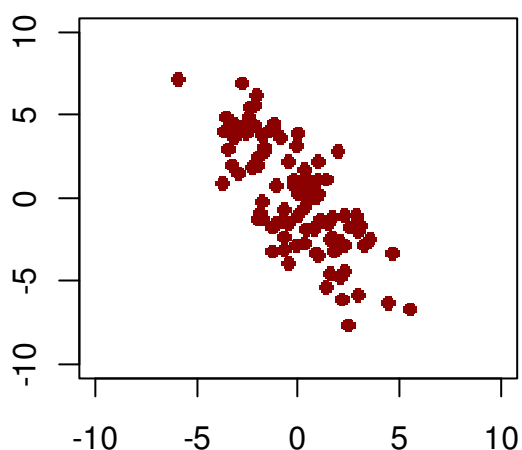
```
ylab='')
plot((T%%a$rotation)[,1],
     (T%%a$rotation)[,2], col='dark
green', pch=16, xlim=c(-10,10),
ylim=c(-10,10), xlab='',
ylab='')
par(op)

# второй рисунок можно получить
проще -
plot(a$x, col='dark blue',
pch=16, xlim=c(-10,10),
ylim=c(-10,10), xlab='',
ylab='')

# первый тоже...
plot(T, col='dark red', pch=16,
xlim=c(-10,10), ylim=c(-10,10),
xlab='', ylab='')

# вывод собственных значений
a = princomp(T)
a$sd

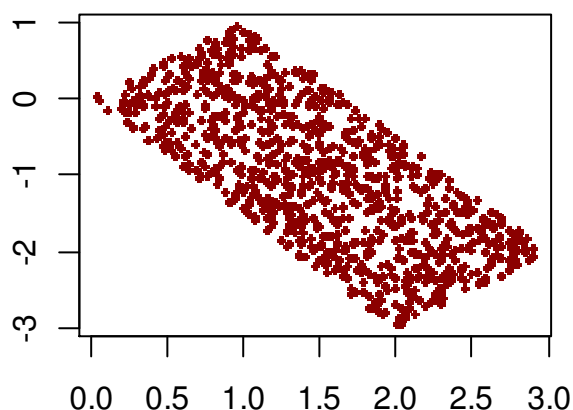
    Comp.1    Comp.2
3.655761  1.257609
```

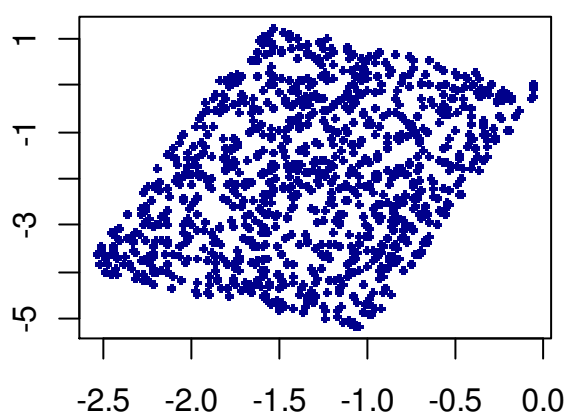
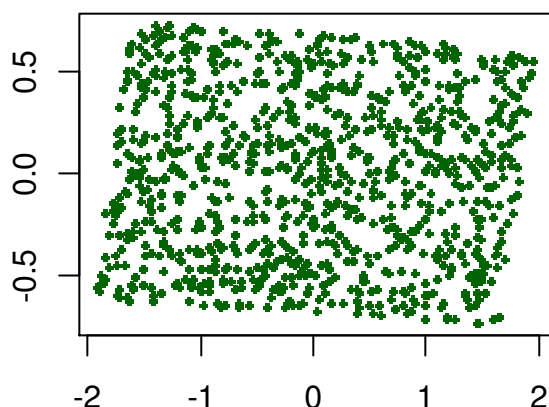


```
library(e1071) # для ICA
# порождение выборки
x = runif(1000);
y = runif(1000);
T = cbind(x+2*y, x-3*y);
# метод PCA
a = prcomp(T)
# метод ICA
b = ica(T, lrate=0.1, ncomp=2)
# рисование
op <- par(mfrow=c(3,1),
mar=c(0,0,0,0)+2)
plot(T, col='dark red', pch=16,
xlab='', ylab='')
plot(a$x, col='dark green',
pch=16, xlab='', ylab='')
plot((T%%b$weights)[,1],
     (T%%b$weights)[,2], col='dark
blue', pch=16, xlab='', ylab='')
par(op)

# в выводе ICA можно проще:
plot(b$projection, col='dark
blue', pch=16, xlab='', ylab='')
```

Сравнение методов главных и независимых компонент.

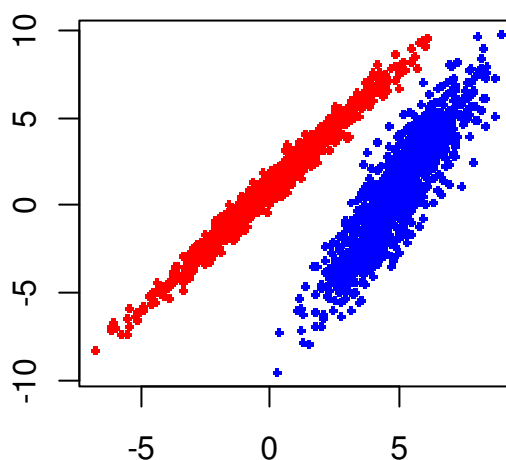


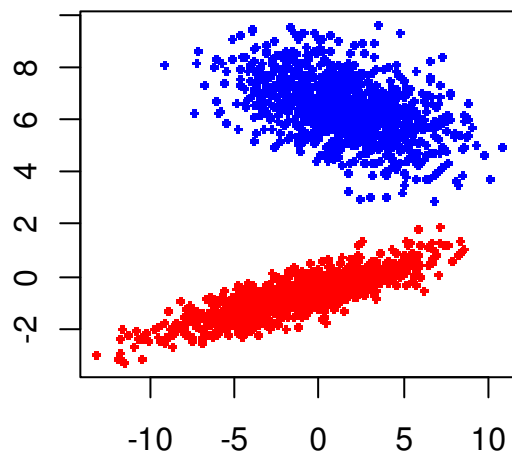


```
library(MASS)
# порождение выборки
x = rnorm(1000);
y = rnorm(1000);
T = rbind(cbind(x+2*y, x+3*y+1),
          cbind(x-y+5, x-3*y+1));
y = c(rep(1,1000), rep(2,1000))
a = prcomp(T) # метод PCA
b = lda(y~T) # метод LDA

cols = c('red', 'blue')
# рисование
op <- par(mfrow=c(2,1),
mar=c(0,0,0,0)+2)
plot(T[,1], T[,2], pch=16,
      xlab='', ylab='', col=cols[y],
      cex=0.5)
plot(a$x[,1], T%*%b$scaling,
      pch=16, xlab='', ylab='',
      col=cols[y], cex=0.5)
par(op)
```

Демонстрация линейного дискриминантного анализа. Вторая картинка получается следующим образом: первая ось – первая главная компонента, вторая – результат LDA (на ней классы максимально хорошо разделяются).





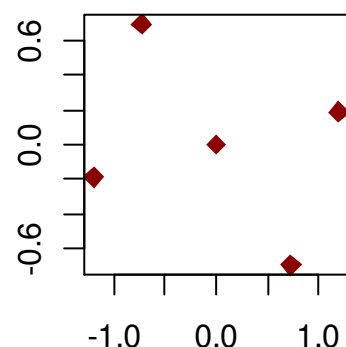
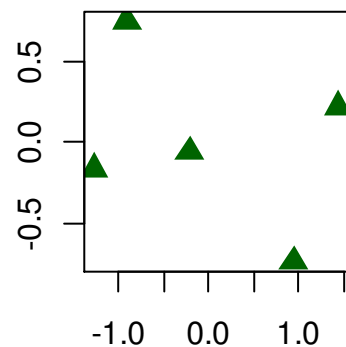
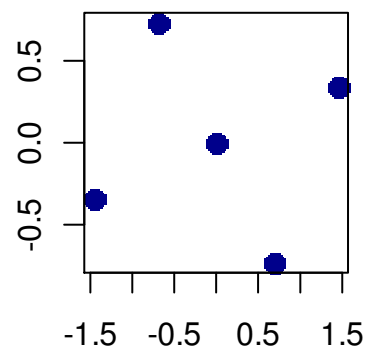
Многомерное шкалирование

```
T = matrix(c(0,1,2,3,1,
1,0,2,2,1, 2,2,0,1,1,
3,2,1,0,1, 1,1,1,1,0), 5)
a = cmdscale(T, k = 2)
plot(a, col='dark blue', pch=16,
xlab='', ylab='', cex=1.5)
T # матрица расстояний
dist(a) # м.р. между
получившимися точками

> dist(a)
      1      2      3      4
2 1.32
3 2.18 2.00
4 3.00 2.18 1.32
5 1.50 1.00 1.00 1.50
> T
      [,1] [,2] [,3] [,4] [,5]
[1,]    0    1    2    3    1
[2,]    1    0    2    2    1
[3,]    2    2    0    1    1
[4,]    3    2    1    0    1
[5,]    1    1    1    1    0
# другой метод
b = isoMDS(T, k=2)
plot(b$points, col='dark green',
pch=17, xlab='', ylab='',
cex=1.5)
dist(b$points)

> dist(b)
      1      2      3      4
2 0.99
3 2.29 2.38
4 2.74 2.41 1.08
5 1.06 1.07 1.34 1.68
```

Необходимо по матрице расстояний изобразить точки на плоскости.



```
# третий метод
c = sammon(T, k=2)
plot(c$points, col='dark red',
     pch=18, xlab='', ylab='',
     cex=1.5)
dist(c$points)

> dist(c)
      1      2      3      4
2 1.00
3 1.98 2.01
4 2.41 1.98 1.00
5 1.21 1.00 1.00 1.21
```

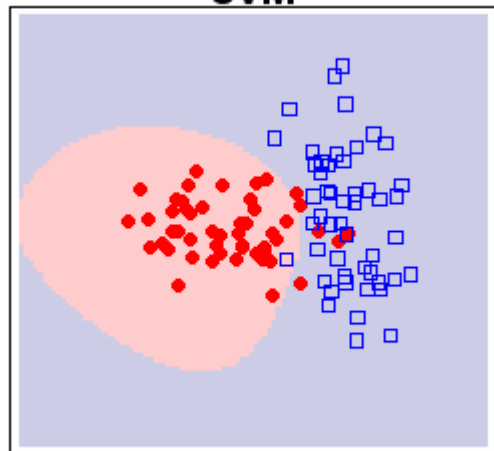
Алгоритмы машинного обучения

```
library(e1071) # для svm
# данные
x = rnorm(50);
y = rnorm(50);
T = rbind(cbind(x+2*y-1, x-y),
          cbind(x-y+4, x+3*y+1));
# классы
y = c(rep(1,50), rep(2,50))
Y <- as.factor(y)
# запуск SVM
r <- svm(T, Y)
# контрольная выборка
n <- 100
x <- cbind( rep(seq(-
10,10,length=n), each=n),
  rep(seq(-10,10,length=n), n) )
# вывод рисунка
z <- predict(r, x) #
классификация
z <- c(rgb(1,.8,.8),
  rgb(.8,.8,.9))[ as.numeric(z) ]
cols = c('red', 'blue')
pchs = c(16,22)
plot(x, col=z, pch=15, xlab="",
     ylab="", main="SVM", axes=FALSE)
box()
points(T, col = cols[y],
       pch=pchs[y])

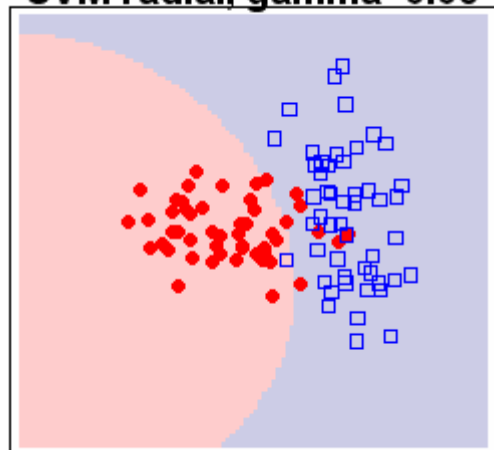
# полиномиальный SVM
r <- svm(T, Y,
  kernel='polynomial')
# вывод рисунка
z <- predict(r, x)
z <- c(rgb(1,.8,.8),
  rgb(.8,.8,.9))[ as.numeric(z) ]
```

Попробуйте самостоятельно получить все приведённые рисунки (Вам придётся проварьировать ядра в методе SVM). Попробуйте также изменять другие параметры метода (например, параметры ядер).

SVM



SVM radial, gamma=0.05



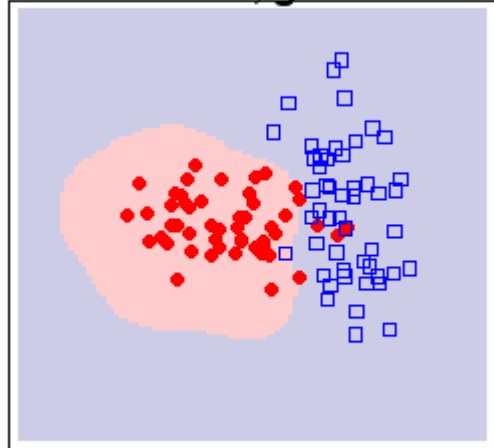

```
op <- par(mar=c(0,0,0,0)+1)
plot(x, col=z, pch=15, xlab="",
ylab="", main="SVM polynomial,
degree=3", axes=FALSE)
box()
points(T, col = cols[y],
pch=pchs[y])
par(op)
```

```
# степень = 2
r <- svm(T, Y,
kernel='polynomial', degree=2)
# вывод рисунка
z <- predict(r, x)
z <- c(rgb(1,.8,.8),
rgb(.8,.8,.9))[ as.numeric(z) ]
op <- par(mar=c(0,0,0,0)+1)
plot(x, col=z, pch=15, xlab="",
ylab="", main="SVM polynomial,
degree=2", axes=FALSE)
box()
points(T, col = cols[y],
pch=pchs[y])
par(op)
```

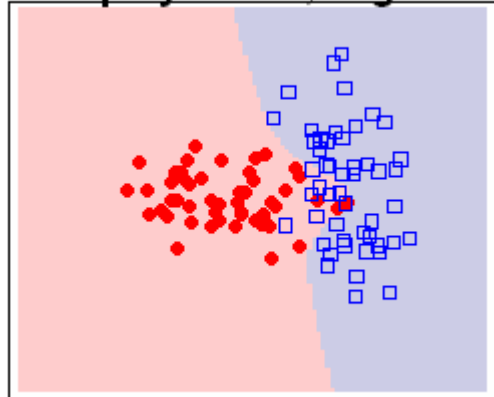
Типичное обращение к функции:

```
model <- svm(x=T, y=Y,
scale=FALSE, type="C-
classification",
kernel="radial", gamma = 3, cost
= 1, class.weights = NULL)
z <- predict(model, x)
```

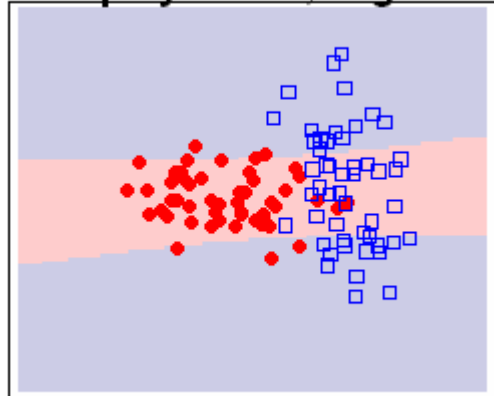
SVM radial, gamma=2



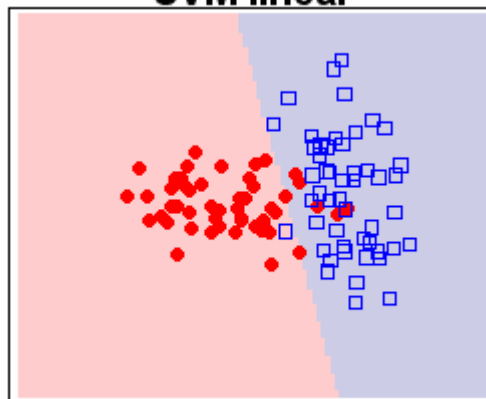
SVM polynomial, degree=3



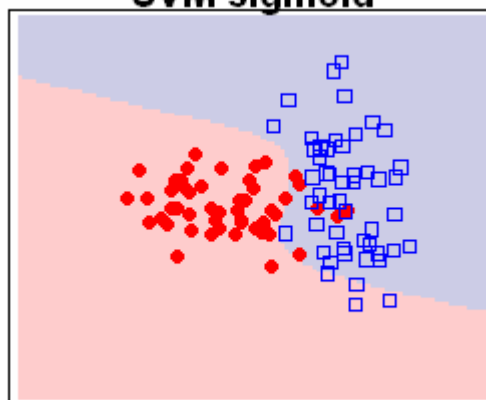
SVM polynomial, degree=2



SVM linear



SVM sigmoid

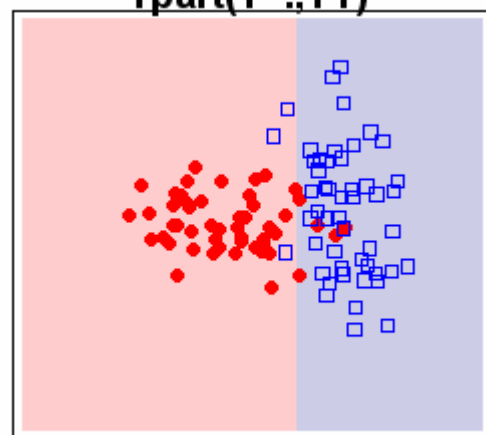


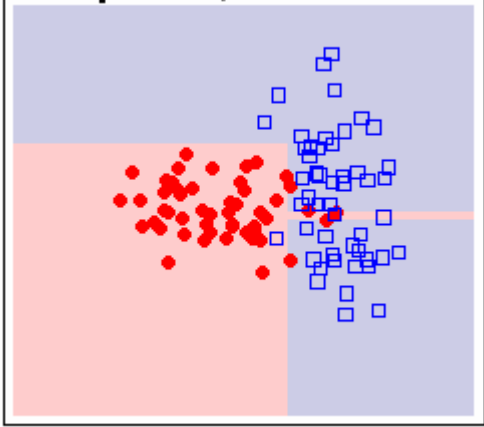
```
# данные из пред. примера

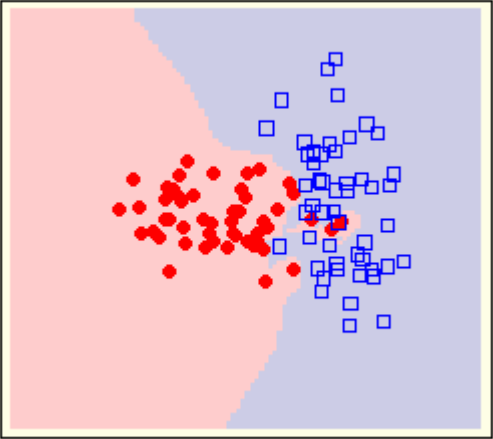
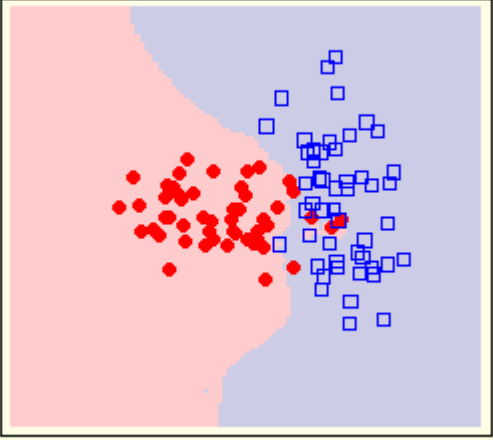
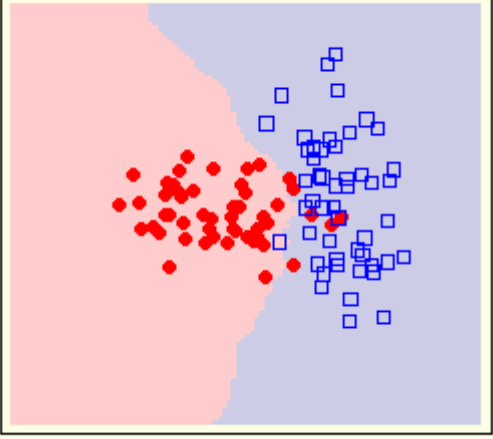
# только такой тип принимает!
YT = data.frame(Y,T)
# построение дерева
r <- rpart(Y~.,YT)
# классификация (тоже
data.frame)
z <- predict(r, data.frame(x))
# преобразование ответа!!!
z <- ifelse(z[,1]>0.5,1,2)
# вывод рисунка
z <- c(rgb(1,.8,.8),
rgb(.8,.8,.9))[ as.numeric(z) ]
op <- par(mar=c(0,0,0,0)+1)
plot(x, col=z, pch=15, xlab="",
ylab="", main="rpart(Y~.,YT)",
axes=FALSE)
box()
points(T, col = cols[y],
pch=pchs[y])
par(op)
# вывод дерева
> r
n= 100
```

Решающее дерево.

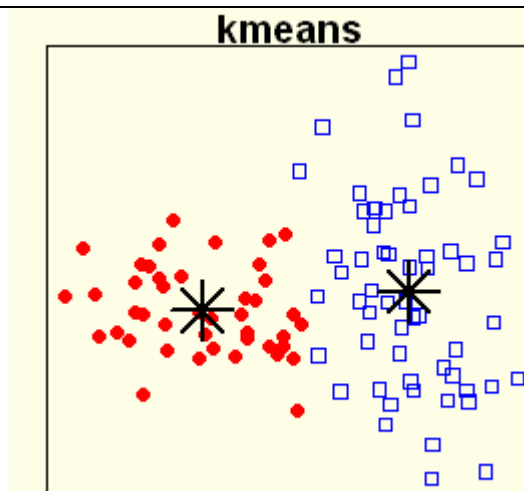
rpart(Y~.,YT)



<pre>node), split, n, loss, yval, (yprob) * denotes terminal node 1) root 100 50 1 (0.5000000 0.5000000) 2) X1< 2.319586 50 3 1 (0.9400000 0.0600000) * 3) X1>=2.319586 50 3 2 (0.0600000 0.9400000) *</pre>	
<pre>r <- rpart(Y~.,YT, control = c(minsplit = 10, minbucket = 2)) n= 100 node), split, n, loss, yval, (yprob) * denotes terminal node 1) root 100 50 1 (0.50000000 0.50000000) 2) X1< 2.319586 50 3 1 (0.94000000 0.06000000) 4) X2< 3.699459 48 1 1 (0.97916667 0.02083333) * 5) X2>=3.699459 2 0 2 (0.00000000 1.00000000) * 3) X1>=2.319586 50 3 2 (0.06000000 0.94000000) 6) X2< 0.1776497 23 3 2 (0.13043478 0.86956522) 12) X2>=-0.1175459 2 0 1 (1.00000000 0.00000000) * 13) X2< -0.1175459 21 1 2 (0.04761905 0.95238095) * 7) X2>=0.1776497 27 0 2 (0.00000000 1.00000000) *</pre>	<p>Показано, как в решающих деревьях задаются параметры (построено «более сложное» дерево).</p> <p>minsplit=10, minbucket=2</p> 
<pre>z <- knn(T, x, Y, k = 5)</pre>	<p>Метод k ближайших соседей (показаны картинки при разных k).</p>

	<div data-bbox="842 181 1417 1720"><p>1NN</p><p>3NN</p><p>5NN</p></div> <p>[КАК РЕАЛИЗОВАН?????]</p>
<pre># данные берем «свм-ные» op <- par(mar=c(0,0,0,0)+1, bg=rgb(1,1,.9)) # поиск центров кластеров cl <- kmeans(T, 2) # рисование plot(T, col=cols[cl\$cluster], pch=pchs[cl\$cluster], xlab="",</pre>	<p>Кластеризация методом k-средних (результат случаен!) Для запуска нужны данные и переменные, определённые при запуске SVM.</p>

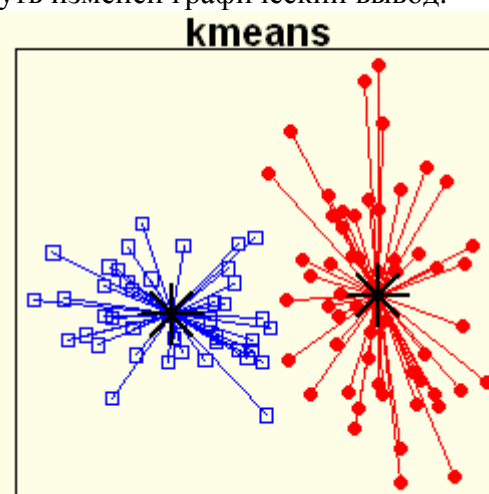
```
ylab="", main="kmeans",
axes=FALSE)
points(cl$centers, col =
'black', pch = 8, cex=3, lw=2)
box()
par(op)
```



```
op <- par(mar=c(0,0,0,0)+1,
bg=rgb(1,1,.9))
# поиск центров кластеров
cl <- kmeans(T, 2)
# рисование
plot(T, col=cols[cl$cluster],
pch=pchs[cl$cluster], xlab="",
ylab="", main="kmeans",
axes=FALSE)
# соединения
segments( T[cl$cluster==1,][,1],
T[cl$cluster==1,][,2],
cl$centers[1,1],
cl$centers[1,2], col=cols[1])
segments( T[cl$cluster==2,][,1],
T[cl$cluster==2,][,2],
cl$centers[2,1],
cl$centers[2,2], col=cols[2])
# центры кластеров
points(cl$centers, col =
'black', pch = 8, cex=3, lw=2)
box()
par(op)
```

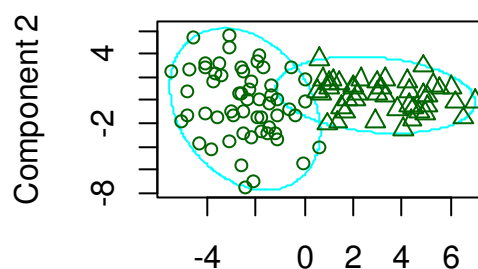
```
clusplot(T, clus=cl$cluster)
```

Чуть изменен графический вывод.



Результат работы «стандартной» функции
 вывода **clusplot**.

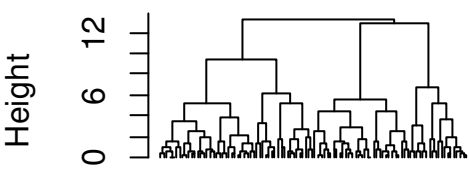
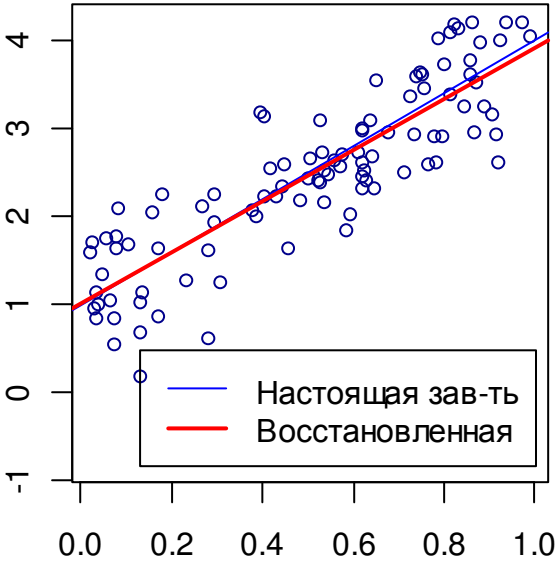
CLUSPLOT(T)



Component 1
 These two components explai

```
# кластеризовать
hc <- hclust(dist(T))
# вывести дендрограмму
plot(hc, labels = FALSE, hang =
-1)
```

Иерархическая кластеризация. Вывод
 дендрограммы.

	<h3 style="text-align: center;">Cluster Dendrogram</h3>  <p style="text-align: center;">dist(T) hclust (*, "complete")</p>
<pre># подготовка данных N <- 100 # число точек a <- 1 b <- 3 epsilon <- rnorm(N, sd=0.5) # шум X <- runif(N) Y <- a + b*X + epsilon # рисование plot(X, Y, col='darkblue', ylim=c(range(Y)[1]-1, range(Y)[2])) abline(a,b, col="blue") abline(lm(Y~X), col="red", lwd=2) # лин. регрессия legend(par('usr')[1]+0.15, par('usr')[3]+1.5, c('Настоящая зав-ть', 'Восстановленная'), lwd=c(1,2), col=c('blue', 'red')) # коэффициент корреляции > cor(X,Y) [1] 0.867497 # корреляция между рангами > cor(rank(X), rank(Y)) [1] 0.8759676 # можно проще... > cor(X,Y, method='spearman') [1] 0.8759676</pre>	<p>Линейная регрессия выполняется командой <code>lm(Y~X)</code>.</p> 
<pre># подготовка данных N <- 20 # число точек</pre>	<p>Три классические регрессии. Тонким пунктиром показаны расстояния, суммы</p>

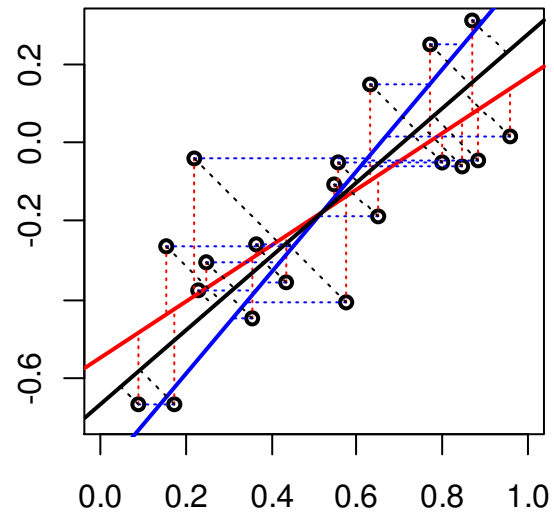
```
a <- -0.5
b <- 0.7
epsilon <- rnorm(N, sd=0.2) # шум
X <-runif(N)
Y <- a + b*X + epsilon
# рисование
plot(X, Y, col='black', lwd=2,
xlim=c(0,1), ylim=c(-0.7,0.3))

# первая регрессия
reg1 = lm(Y~X)
abline(reg1, col="red", lwd=2) # лин. регрессия
segments(X, Y, X,
reg1$fitted.values,col="red",
lty=3)

# вторая регрессия
reg2 = lm(X~Y)
abline(-reg2$coefficients[1]/
reg2$coefficients[2],
1/reg2$coefficients[2],
col="blue", lwd=2) # лин.
регрессия
segments(X, Y,
reg2$fitted.values,
Y,col="blue", lty=3)

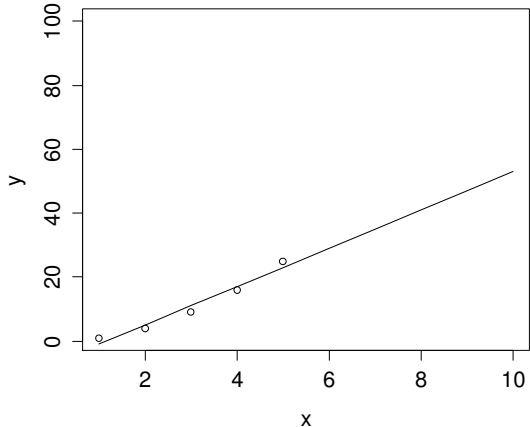
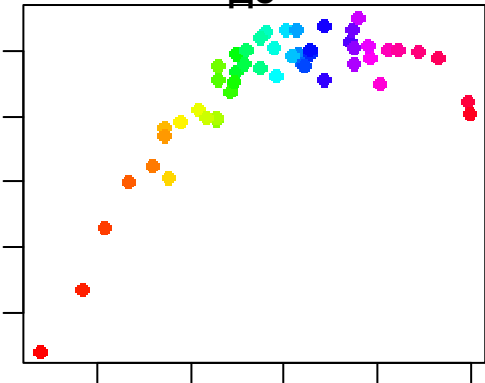
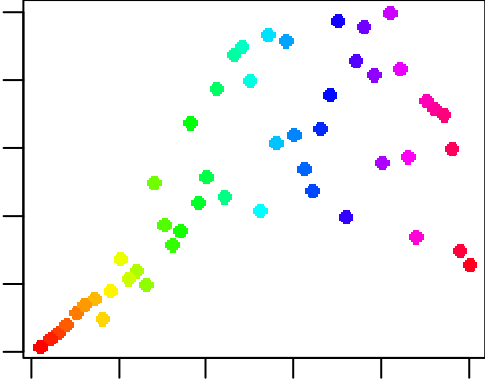
# третья регрессия
XY<-cbind(X,Y)
reg3 <- princomp(XY)
b <- reg3$loadings[2,1] /
reg3$loadings[1,1]
a <- reg3$center[2] - b *
reg3$center[1]
abline(a, b, col="black", lwd=2)
XY <- reg3$center +
outer(reg3$loadings[,1],
(solve(reg3$loadings, (t(XY) -
reg3$center)) [1,]))
segments( X, Y, XY[1,], XY[2,],
col="black", lty=3)
```

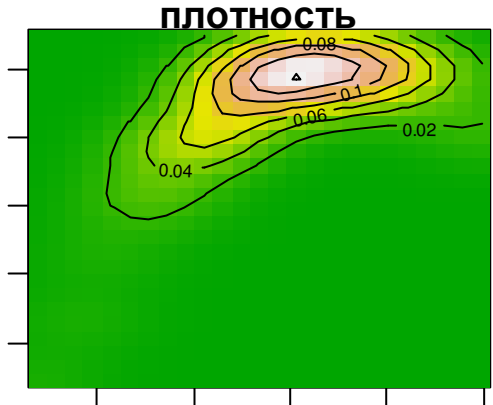
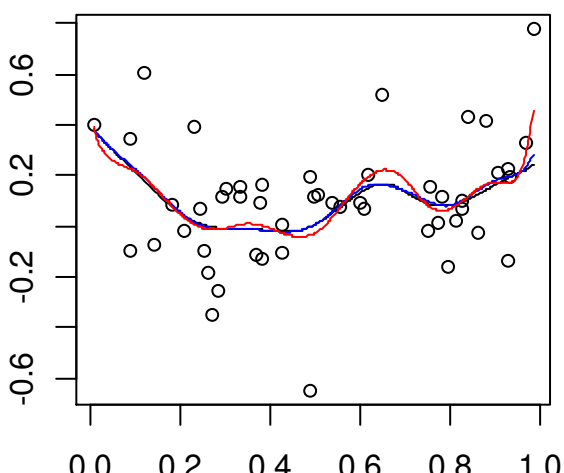
которых они минимизируют.



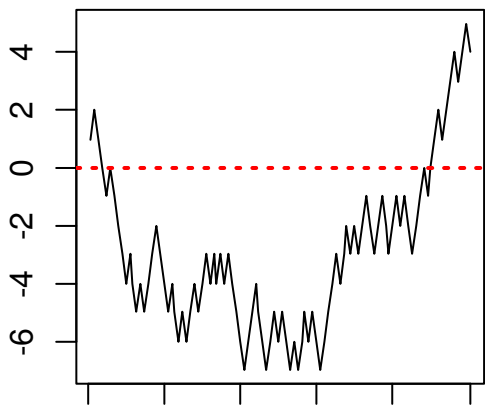
```
x <-c(1,2,3,4,5)
y <-x^2
r<-lm(y~x)
# ТОНКОСТЬ
df = as.data.frame(1:10)
colnames(df) = 'x'
# чтобы это работало
z<-predict(r, newdata=df)
plot(y~x, xlim=c(1,10),
```

Вроде, функция **predict** для линейной регрессии работает только так... на обучающей выборке.

<pre>ylim=c(1,100)) lines(df[, 'x'], z)</pre>	
<pre>summary(lm(y~I((x1+x2)^2))) summary(lm(y~poly((x1+x2), 2)))</pre>	<p>Если сделать $y \sim (x1+x2)^2$, то это эквивалентно $y \sim (x1+x2+x1:x2)$. [Подробнее]</p>
<pre># данные x <- rnorm(50) y <- rnorm(50, sd=0.5) + x - x^2 # рисование op <- par(mfrow=c(2,1)) cols = rainbow(50)[rank(x)] # сами данные plot(x,y, labels=FALSE, xlab="", ylab="", col = cols, pch=16, main="до") # после преобразования plot(rank(x), rank(y), labels=FALSE, xlab="", ylab="", col=cols, pch=16, main="после") par(op) # построение 2D-плотности library(MASS) op <- par(mar=c(0,0,0,0)+1) r <- kde2d(x, y) image(r, col = terrain.colors(1000), main='плотность') contour(kde2d(x,y), lwd=1, add=TRUE) par(op)</pre>	<p>Приём, который используется в анализе данных – замена значений на «ранг» (порядковый номер).</p> <div style="text-align: center;"> <p>до</p>  <p>после</p>  </div> <p>Теперь изображаем плотность исходной выборки.</p>

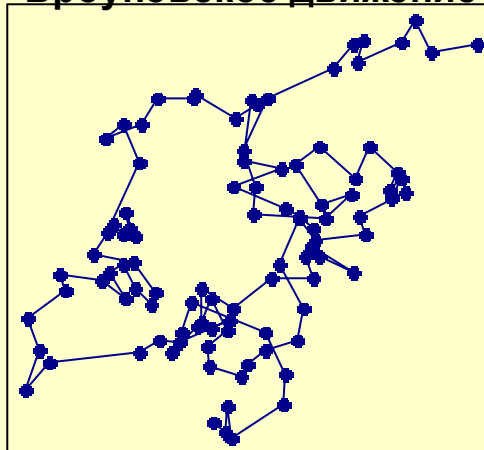
	
<pre># данные x <- runif(50) y <- (x-0.5)^2+rnorm(50, sd=0.2) plot(y~x) # л.п.регрессия fit2 <- locpoly(x, y, degree=2, bandwidth = 0.1) lines(fit2) fit3 <- locpoly(x, y, degree=3, bandwidth = 0.1) lines(fit3, col='blue') fit4 <- locpoly(x, y, degree=4, bandwidth = 0.1) lines(fit4, col='red')</pre>	<p>Локальная полиномиальная регрессия.</p> 

Распределения

<pre>n <- 100 # длина выборки x = sample(c(-1,1), n, replace=TRUE) # с.в. из {-1,+1} plot(cumsum(x), type='l', main='случайное блуждание') abline(h=0, lty=3, col='red') # линия</pre>	<p>Демонстрация случайного блуждания.</p> <p>случайное блуждание</p>  <p>Попробуйте породить выборку так:</p> <pre>x = sample(c(-1,1), n, replace=TRUE, prob=c(0.3, 0.7))</pre>
---	--

```
op <- par(bg=rgb(1, 1, 0.8),
mar=c(0,0,0,0)+1)
N <- 100 # число шагов
x <- cumsum(rnorm(N))
y <- cumsum(rnorm(N))
plot(x, y,
type = "o", pch = 16, lwd = 1,
xlab = "", ylab = "",
axes = FALSE,
main = "Броуновское движение",
col='darkblue')
box()
par(op)
```

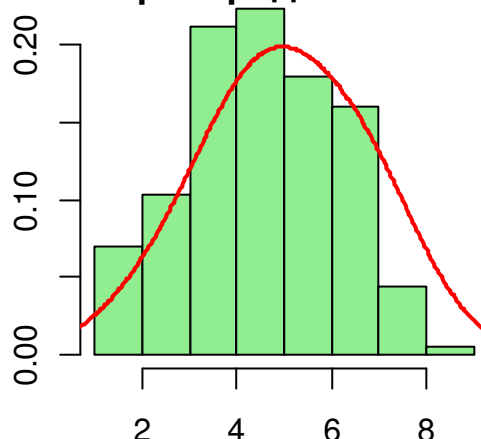
Броуновское движение



```
N <- 500 # длина выборки
n <- 10 # число "блоков"
p <- 0.5
x <- rbinom(N,n,p) # генерация
с.в.
# вывод картинки
hist(x, xlim = c(min(x),
max(x)), probability = TRUE,
nclass = max(x) - min(x) + 1,
col = 'lightgreen', main =
'Биномиальное\нраспределение')
lines(density(x, bw=1), col =
'red', lwd = 2)
```

Демонстрация биномиального распределения. Обратите внимание, как название сделать «в две строчки». Выполните эти команды с большими значениями **N** и **n**.

Биномиальное распределение



Для генерации выборки можно было использовать команды

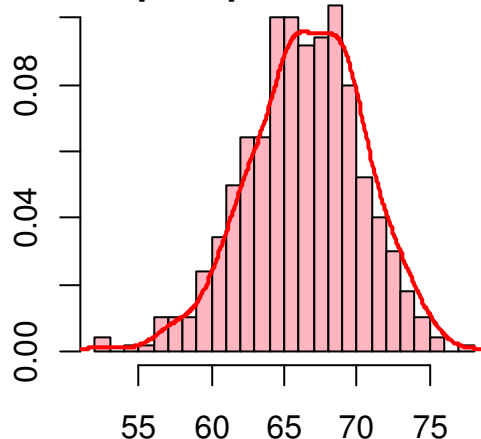
```
x <- rep(NA, N)
for (i in 1:N) x[i] <- sum(
sample( c(1,0), n, replace =
TRUE, prob = c(p, 1-p) ))
```

```
N <- 500
x <- rhyper(N, 200, 100, 100) #
генерация с.в.
# вывод картинки
hist(x, xlim = c(min(x),
max(x)), probability = TRUE,
nclass = max(x) - min(x) + 1,
col = 'lightpink', main =
'Гипергеометрическое\н
распределение')
lines(density(x, bw=1), col =
```

Аналогично, гипергеометрическое распределение

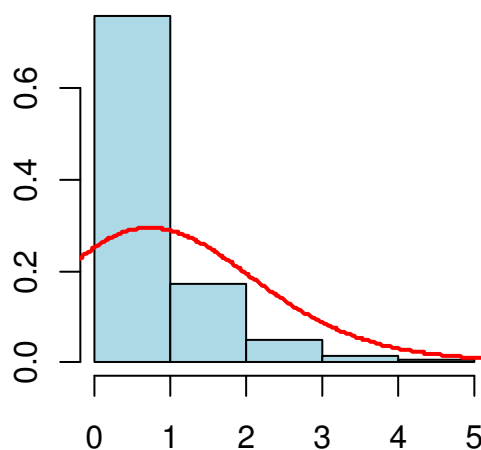
```
'red', lwd = 2)
```

Гипергеометрическое распределение

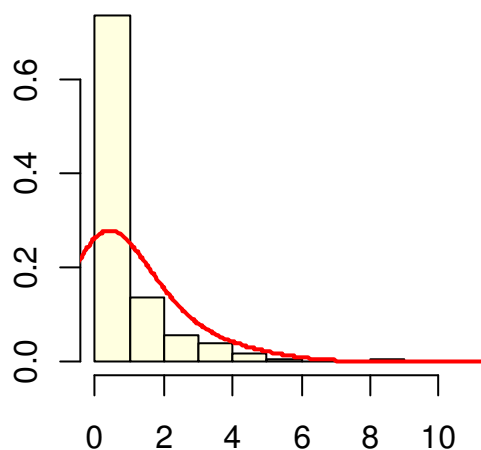


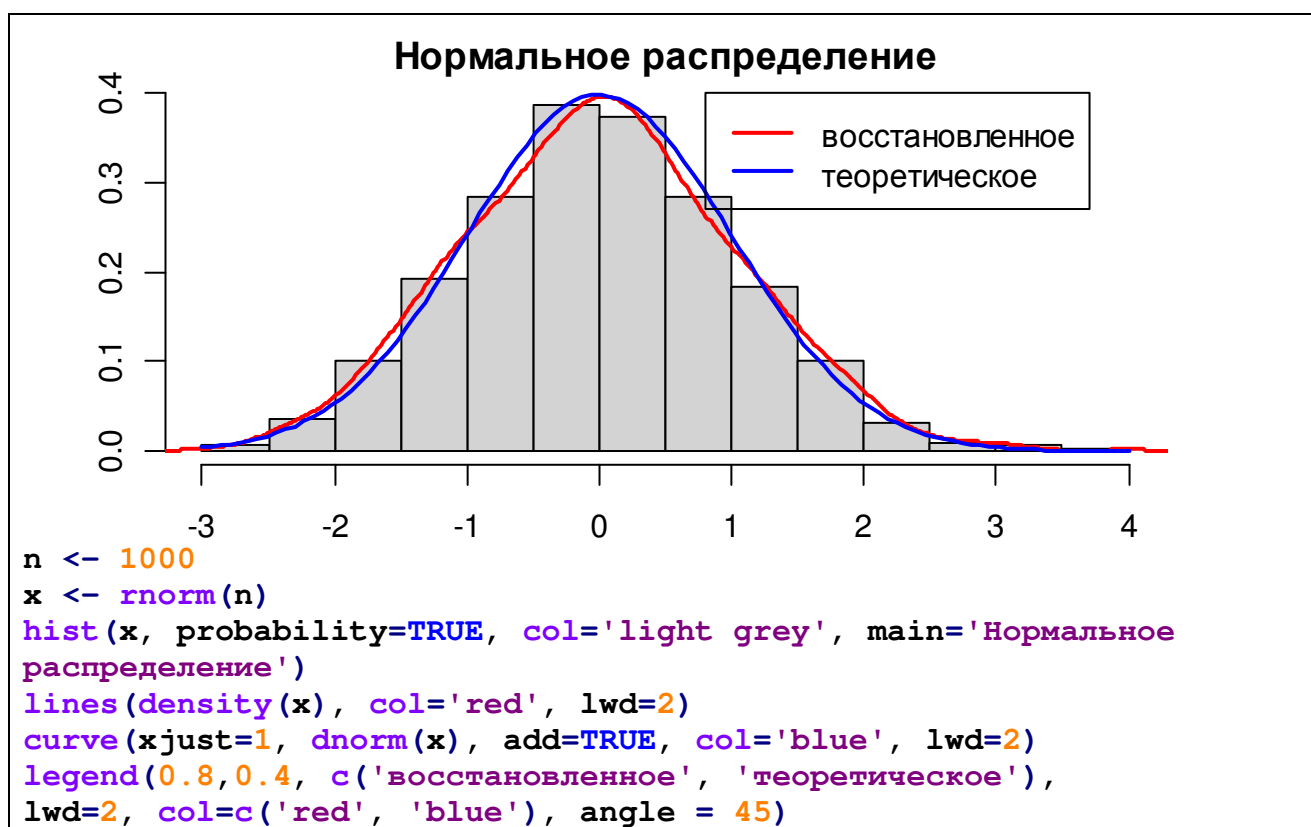
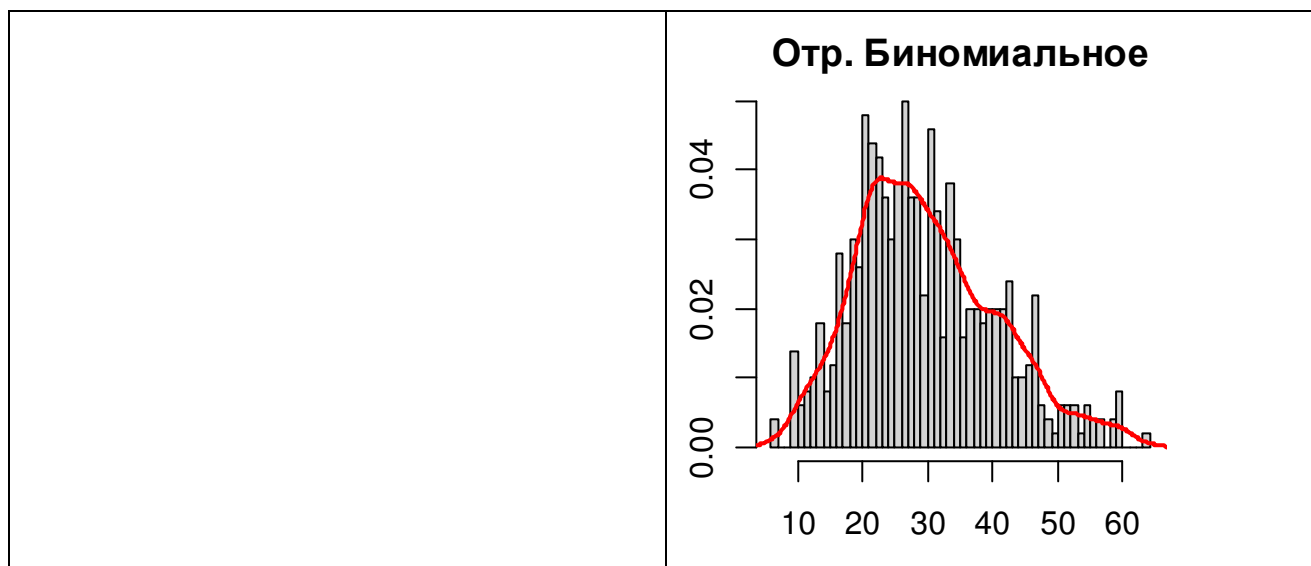
```
N <- 500
x <- rpois(N, 1)
# вывод картинки
hist(x, xlim = c(min(x),
max(x)), probability = TRUE,
nclass = max(x) - min(x) + 1,
col = 'lightblue', main =
'Распределение Пуассона')
lines(density(x, bw=1), col =
'red', lwd = 2)
```

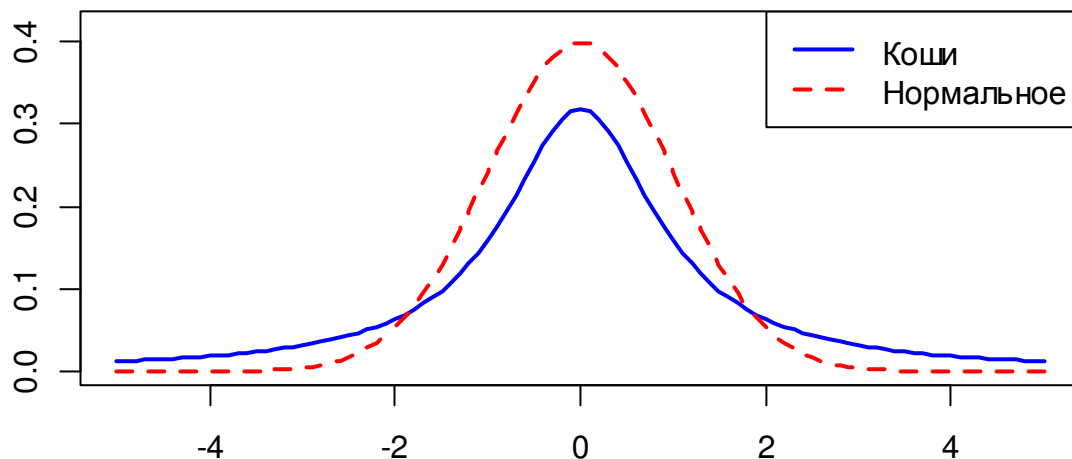
Распределение Пуассона



Геометрическое

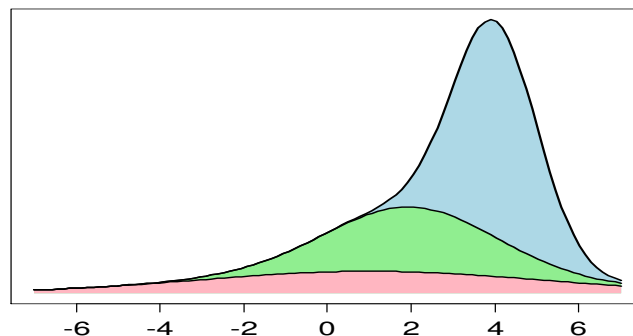






```
curve(dcauchy(x), xlim=c(-5,5), ylim=c(0,0.42), col='blue', lwd=2)
curve(dnorm(x), add=TRUE, col='red', lwd=2, lty=2)
legend(par('usr')[2], par('usr')[4], xjust=1,
c('Коши', 'Нормальное'),
lwd=c(2,2),
lty=c(1,2),
col=c('blue', 'red'))
```

Смесь норм. распределений



```
n <- 100
x <- seq(-7, 7, length=n)
p <- c(0.2, 0.3, 0.5)
m <- c(1, 2, 4)
s <- c(4, 2, 1)
y3 <- p[1] * dnorm(x, mean=m[1], sd=s[1]) +
p[2] * dnorm(x, mean=m[2], sd=s[2]) +
p[3] * dnorm(x, mean=m[3], sd=s[3])
y2 <- p[1] * dnorm(x, mean=m[1], sd=s[1]) +
p[2] * dnorm(x, mean=m[2], sd=s[2])
y1 <- p[1] * dnorm(x, mean=m[1], sd=s[1])
plot.new()
plot.window(xlim=range(x), ylim=range(0, y1, y2, y3), main="M")
polygon(c(x[1], x, x[n]), c(0, y3, 0), col="lightblue", border=NA)
polygon(c(x[1], x, x[n]), c(0, y2, 0), col="lightgreen", border=NA)
polygon(c(x[1], x, x[n]), c(0, y1, 0), col="lightpink", border=NA)
lines(x, y1, lwd=2)
lines(x, y2, lwd=2)
```

```
lines(x, y3, lwd=2)
box()
axis(1)
title("Смесь норм. распределений")
```

Отладка...

<pre>> x <- 2 > stopifnot(x>0) > x <- -2 > stopifnot(x>0) Error: x > 0 не TRUE</pre>	<p>Прекращение выполнения при определённых условиях.</p> <p>Кстати, функция <code>traceback()</code> после ошибки перечисляет стек вызывавшихся функций.</p>
<pre>> f <- function(x) { x = x+1 x = x/2 x = x-1 } > debug(f) > f(5) debugging in: f(5) debug at #2: { x = x + 1 x = x/2 x = x - 1 } Browse[2]> x # чему равен x? [1] 5 Browse[2]> n # дальше debug at #3: x = x + 1 Browse[2]> where where 1: f(5) Browse[2]> c # продолжить exiting from: f(5)</pre>	<p>Отладка функции.</p> <p><code>n = next</code> <code>c = continue</code></p>
<pre>> h <- function(x) { x = x+1 return(x) } > trace(h, browser) [1] "h" > h(2) Tracing h(2) on entry Called from: eval(expr, envir, enclos) Browse[1]> c [1] 3 > untrace(h) > h(2) [1] 3</pre>	<p>Другой способ отладки.</p> <p>Указываем, какую функцию вызывать при вызове функции <code>h</code>.</p>
<pre>> g <- function(x)</pre>	<p>«Условная отладка».</p>

<pre>{ x = x+1 if (x<0) browser() x = x/2 x = x-1 } > g(2) > g(-2) Called from: g(-2) Browse[1]> c</pre>	
<pre>setBreakpoint("x.R", 28)</pre>	Установка брейкпоинта.
<pre>> runif(3) [1] 0.913602933 0.009119434 0.158165920 > set.seed(516) > runif(3) [1] 0.1554475 0.5151574 0.1704808 > set.seed(516) > runif(3) [1] 0.1554475 0.5151574 0.1704808</pre>	Совет: если используете псевдослучайные сила (а многие функции, например gbm, их используют), то всегда устанавливайте генератор , чтобы иметь возможность повторить эксперимент при тех же условиях, при которых возникла ошибка.
	Есть также различные пакеты для отладки: - debug - edtdbg (для Vim, Emacs)

Пакеты (библиотеки)

<pre>install.packages('name', dependencies=TRUE)</pre>	Установка пакета с именем name (производится через интернет). dependencies=TRUE означает, что дополнительно устанавливаются все пакеты, от которых зависит работа пакета name .
<pre>> library('gbm') Загрузка требуемого пакета: survival Загрузка требуемого пакета: splines Загрузка требуемого пакета: lattice Загрузка требуемого пакета: parallel Loaded gbm 2.1</pre>	Для того, чтобы пользоваться пакетом, его надо предварительно «подключить» (загрузить в память).
<pre>> path.package() "C:/R-3.0.1/library/gbm" "C:/R-3.0.1/library/parallel" "C:/R-3.0.1/library/lattice" "C:/R-30~1.1/library/base"</pre>	Вывод загруженных пакетов
<pre>> .libPaths() [1] "C:/R-3.0.1/library"</pre>	Список директорий, в которых R ищет пакеты для загрузки.

	Если вызвать с аргументом – можно внести изменение в этот список.
<code>> update.packages()</code>	Обновить пакеты.
<code>> data()</code>	Доступ к данным из загруженных пакетов.

Параллельные вычисления

<pre># вычисление суммы множества # строк suma <- function(irows, M) { s <- 0 for (i in irows) for (j in 1:ncol(M)) { s <- s + M[i,j] } return(s) } library(snow) # два процесса cl <- makeCluster(type="SOCK", c("localhost", "localhost")) # матрица с данными M <- matrix(sample(0:1, 1000000, replace=TRUE), nrow=1000) # разбиваем строки на 2 группы irows <- split(1:nrow(M), 1:length(cl)) # отдаём их процессам sums <- clusterApply(cl, irows, suma, M) # "объединяем ответы" print(do.call(sum, sums)) # обратите внимание sum(sums) не # работает</pre>	<p>Пример использования пакета snow.</p> <p>Естественно, это «игровой» пример. Аналогичную работу делает функция sum.</p>
---	---

gputools

книга: <http://heather.cs.ucdavis.edu/parprocbook>

Разное...

<pre>> options(warn=-1) > split(1:10, 1:3) > options(warn=0) \$`1` [1] 1 4 7 10 \$`2` [1] 2 5 8 \$`3` [1] 3 6 9</pre>	Предотвращение вывода предупреждений.
--	---------------------------------------

<pre>> expand.grid(c(1,2), c(3,4)) Var1 Var2 1 1 3 2 2 3 3 1 4 4 2 4</pre>	Декартово произведение.
<pre>> match(c(1,2,3), c(2,3,4)) [1] NA 1 2 > x [1] 1 4 2 3 2 3 5 5 5 2 > match(x, c(2,4,1), nomatch=0) [1] 3 2 1 0 1 0 0 0 0 1</pre>	Функция соответствия.
<pre>substitute(x+x+1-y, list(x=2,y=3)) 2 + 2 + 1 - 3</pre>	Подстановка...
<pre>> s = '1+2' > eval(s) [1] "1+2" > eval(parse(text=s)) [1] 3</pre>	parse – перевод строки в «вычисляемое выражение», eval – вычисление.
<pre>> exists('T') [1] TRUE > exists('a') [1] FALSE > a = 2i > exists('a') [1] TRUE</pre>	Наличие переменной проверяется функцией. Напомним, что T=TRUE .

Создание своего пакета

<pre>> f <- function(x,y) {x+y} > g <- function(x,y) {x-y} > a = 10.01 > package.skeleton("mypackage", c("f", "g", "a"))</pre>	Создается директория mypackage .
--	---

Справочник по командам R для знатоков MatLab-a

* html R for MATLAB users (вроде: Vidar Bronken Gundersen, /mathesaurus.sf.net)
 David Hiebeler MATLAB/R Reference <http://www.math.umaine.edu/~hiebler>

Далее покажем соответствия команд R и системы Matlab. Для краткости указываем только те функции и операции, в которых есть отличия.

R-команда	M-команда
Работа с системой	
help.start()	doc Вызов справки (в R – гипертекстовой).
help(plot) или ?plot Обратите внимание на команду help(Syntax)	help plot Помощь по функции.

<code>help(package='splines')</code>	<code>help splines</code> или <code>doc splines</code> Помощь по пакету [БИБЛИОТЕКЕ].
<code>demo()</code>	<code>demo</code> Демонстрация.
<code>example(plot)</code>	Примеры использования функции (в MATLABе нет).
<code>source('foo.R')</code>	<code>foo</code> или <code>foo.m</code> Запустить содержимое файла.
В R нет похожей функциональности!	<code>which sqrt</code> Показывает путь к файлу, в котором определена функция <code>sqrt</code> .
<code>history()</code>	<code>history</code> История команд
<code>savehistory(file=".Rhistory")</code>	<code>diary on</code> ... <code>diary off</code> Запись истории команд в файл.
<code>q(save='no')</code>	<code>exit</code> или <code>quit</code> Закончить работу.
<code>library(RSvgDevice)</code> Перед подключением надо установить библиотеку <code>install.packages('RSvgDevice')</code>	Подключение библиотеки. В системе MATLAB достаточно воспользоваться опцией SETPATH.
<code># комментарий в R</code>	<code>% комментарий в MATLAB</code>
<code>x <- 1 + 2</code>	<code>x = 1 + ... 2</code> Разрыв строки.
<code>.Last.value</code>	<code>ans</code> «Последнее вычисленное значение».
<code>objects()</code> <code>ls.str(pattern='ab')</code>	<code>who</code> или <code>whos</code> Список переменных в памяти <code>whos *ab*</code> Список переменных, в имена которых входит последовательность букв 'ab'.
<code>rm(x)</code> <code>rm(list=ls())</code>	<code>clear x</code> Удаление из памяти. <code>clear all</code> Удаление из памяти всех переменных.
<code>list.files()</code> или <code>dir()</code>	<code>dir</code> или <code>ls</code> Содержимое текущей директории.
<code>list.files(pattern=".r\$")</code>	<code>what</code> Файлы с кодом в текущей директории.
<code>getwd()</code> <code>setwd('foo')</code>	<code>pwd</code> <code>cd foo</code>

	Текущая директория и её изменение.
<code>save.image(file='foo.rda')</code> <code>load('foo.rda')</code>	<code>save foo.mat</code> <code>load foo.mat</code> Сохранение всех переменных в файл / загрузка из файла.
<code>write(t(A), file='c:\\data.txt', ncolumn=dim(A)[2]) # t(A)!!!</code> <code>B = as.matrix(read.table('c:\\data.txt', skip=s))</code>	<code>save data.txt A -ascii</code> <code>tmp = importdata('data.txt', ' ' , s);</code> <code>B = tmp.data</code> Сохранение и загрузка данных в таблицу текстового формата.
<code>system("notepad")</code>	<code>!notepad</code> <code>system("notepad")</code> Вызов системных команд.
<code>set.seed(10)</code>	<code>rand('state', 10)</code> Установка генератора псевдослучайных чисел в фиксированное состояние.
<code>options(digits=6) # рекомендация</code>	<code>format short g and</code> <code>format long g</code> Контроль формата вывода.
<code>Sys.sleep(x)</code>	<code>pause(x)</code> Пауза x секунд.
<code>.Machine\$double.eps</code>	<code>eps</code> Машинная точность.
<code>t1 = proc.time()</code> <code>proc.time()-t1</code>	<code>t1=cputime; cputime-t1 % CPU- time</code> <code>tic; toc</code> Засечка времени и просмотр, сколько времени прошло.
<code>stop('message')</code> <code>warning('message')</code>	<code>error('message')</code> Остановка вычислений с выдачей сообщения об ошибке. <code>warning('message')</code> Выдача предупреждения.
Операции	
<code>a <- 1</code>	<code>a=1;</code> Присваивание.
<code>1i</code> <code>Re(a)</code> <code>Im(a)</code> <code>Arg(a)</code> <code>Conj(a)</code>	<code>i</code> или <code>j</code> Мнимая единица. <code>real(a)</code> <code>imag(a)</code> <code>arg(a)</code> <code>conj(a)</code>
<code>a ^ b</code>	<code>a.^b;</code> Возведение в степень.
<code>a * a</code>	<code>a.*a</code> Поэлементное умножение.
<code>a %% b</code>	<code>rem(a,b)</code> или <code>mod(a,b)</code> Остаток от деления.

<code>a %/% b</code>	Целочисленное деление (нет в MATLAB-e).
<code>a != b</code> <code>!x</code>	<code>a ~= b</code> Неравно. <code>~x</code> или <code>not (x)</code> Не x.
<code>ceiling(x)</code>	<code>ceil(x)</code> Округление к ближайшему целому, которое больше данного.
	Отбрасывание дробной части <code>fix(a)</code> не имеет аналогов в R.
<code>sum(a*b)</code>	<code>dot(a,b)</code> Скалярное произведение векторов.
<code>2*pnorm(x*sqrt(2))-1</code> <code>qnorm((1+x)/2)/sqrt(2)</code>	<code>erf(x)</code> <code>erfinv(y)</code> Функция ошибки (и её обратная).
<code>y <- 7</code> <code>get('y', envir=globalenv())</code> <code># если нет локальной пер. y</code> <code>fix(A)</code>	<code>assignin('base', 'y', 7)</code> <code>evalin('base', 'y')</code> Работа с глобальными переменными.
	<code>openvar(A)</code> Отредактировать значение переменной. Вызывается редактор для редактирования!
Теоретико-множественные операции	
<code>setdiff(union(a,b), intersect(a,b))</code>	<code>setxor(a,b)</code> Симметрическая разность
<code>is.element(2,a)</code> или <code>2 %in% a</code>	<code>ismember(2,a)</code> «Входит во множество».
<code>choose(10,3)</code>	<code>nchoosek(10,3)</code> Число сочетаний.
Порождения	
<code>a <- c(2,3,4,5)</code> <code>b <- t(t(c(2,3,4,5)))</code> <code># или</code> <code>b <- as.matrix(c(2,3,4,5), ncol=1)</code> В R нет ориентации вектора! Можно упростить ввод с помощью функции <code>a = scan()</code>	<code>a = [2 3 4 5]</code> <code>b = [2 3 4 5]'</code> Можно упростить ввод с помощью <code>input('')</code>
<code>rnorm(10)</code> <code>runif(10, min=1, max=5)</code> <code>matrix(runif(36), 6)</code>	<code>randn(1,10)</code> <code>1+4*rand(1,10)</code> <code>rand(6)</code> Случайные векторы.
<code>seq(10)</code> или <code>1:10</code>	<code>1:10</code>
<code>seq(10,1)</code> или <code>10:1</code> Обратите внимание: не указываем «-1». <code>seq(from=10,to=1,by=-3)</code>	<code>10:-1:1</code> <code>10:-3:1</code>

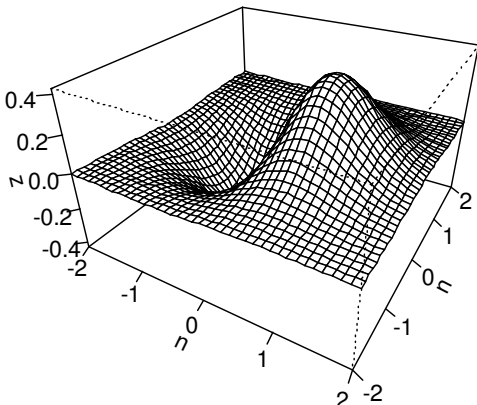
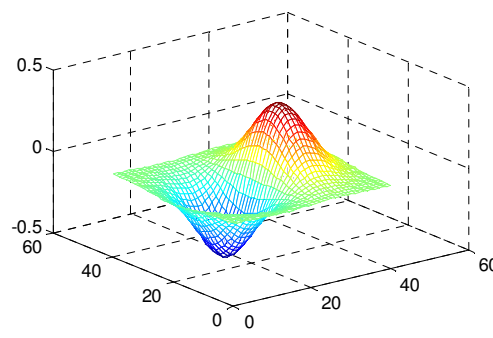
<code>seq(1, 10, length=7)</code>	<code>linspace(1, 10, 7)</code>
<code>10^seq(a, b, len=n)</code>	<code>logspace(a, b, n)</code>
<code>rev(a)</code>	<code>fliplr(a)</code> или <code>a(end:-1:1)</code>
<code>b[]=2</code>	<code>a(:) = 3</code>
<code>c(a, a)</code>	<code>[a a]</code> Конкатенация.
<code>a <- 1:3</code> <code>rep(a, 3)</code>	<code>a = 1:3</code> <code>repmat(a, 1, 3)</code> 1 2 3 1 2 3 1 2 3
<code>a <- 1:3</code> <code>rep(a, each=2)</code>	<code>a = 1:3</code> <code>a = repmat(a, 2, 1)</code> <code>a = a(:)'</code> 1 1 2 2 3 3
<code>a <- 1:3</code> <code>rep(a, a)</code>	[сложнее] 1 2 2 3 3 3
<code>a[-1]</code>	<code>a(2:end)</code> Без первого элемента.
<code>a[-i]</code>	<code>a([1:i-1 i+1:end])</code> Не очень эффективно.
<code>a[length(a)]</code> Нет короткой конструкции.	<code>a(end)</code> Последний элемент.
<code>a = a[1:3]</code> или <code>length(a) <- 3</code>	<code>a = a(1:3)</code> Обрезка вектора.
<code>sample(k, 1)</code>	<code>floor(k*rand) + 1</code> Случайное целое число от 1 до k.
<code>sample(n)</code>	<code>randperm(n)</code> Случайная перестановка.
Функции над векторами и матрицами	
<code>pmax(a, b)</code> <code>max(a, b)</code> <code>v <- max(a); i <- which.max(a)</code>	<code>max(a, b)</code> Поэлементный максимум. <code>max([a b])</code> Максимум из элементов 2х векторов. <code>[v, i] = max(a)</code> Стандартная функция максимума в MATLABe.
<code>i <- order(a)</code>	<code>[~, i] = sort(a(:))</code> Индексы элементов (для сортировки).
<code>tmp=sort(v, index.return=TRUE);</code> <code>s=tmp\$x; idx=tmp\$ixs</code>	<code>[s, idx]=sort(v)</code> Сортировка элементов и возврат индексов.
Порождение матриц	
<code>rbind(c(2, 3), c(4, 5))</code> <code>matrix(c(2, 3, 4, 5), nrow=2, byrow=TRUE)</code> <code>array(c(2, 4, 3, 5), dim=c(2, 2))</code>	<code>a = [2 3; 4 5]</code> Породить матрицу 2 3 4 5
<code>a <- 1:3</code> <code>b <- 5:7</code> <code>rbind(a, b)</code> <code>cbind(a, b)</code>	<code>a = 1:3</code> <code>b = 5:7</code> <code>[a; b]</code> <code>[a', b']</code> Получение матриц 1 2 3 5 6 7 и 1 5

	2 6 3 7
<code>matrix(0,3,5)</code> или <code>array(0,c(3,5))</code> <code>matrix(1,3,5)</code> или <code>array(1,c(3,5))</code> <code>matrix(9,3,5)</code> или <code>array(9,c(3,5))</code> <code>diag(1,3)</code> <code>diag(c(4,5,6))</code>	<code>zeros(3,5)</code> <code>ones(3,5)</code> <code>ones(3,5)*9</code> или <code>repmat(9,3,5)</code> <code>eye(3)</code> <code>diag([4 5 6])</code> Получение особых матриц (из всех нулей, единиц, констант, единичной и диагональной).
<code>matrix(runif(m*n),m,n)</code>	<code>rand(m,n)</code> Порождение случайной матрицы.
	<code>magic(3)</code>
<code>matrix(1:6,nrow=3,byrow=T)</code> <code>matrix(1:6,nrow=2)</code> или <code>array(1:6,c(2,3))</code>	<code>reshape(1:6,3,2)'</code> <code>reshape(1:6,2,3)</code> Изменение размеров.
<code>> c(x)</code> <code>[1] 2 1 5 3</code> <code>> as.vector(x)</code> <code>[1] 2 1 5 3</code>	<code>x(:)</code> Векторизация.
<code>a[row(a) <= col(a)]</code>	Элементы не ниже главной диагонали. Нет такого! Что-то подобное делает <code>triu(a)</code>
Операции с матрицами	
<code>a[2,3] <- 2</code> <code>a[1,]</code> <code>a[,1]</code> или <code>a[,1,drop=FALSE]</code> , чтобы результат был подматрицей <code>a(c(1,3), c(1,4))</code>	<code>a(2,3)=2</code> Элемент. <code>a(1,:)</code> Строка. <code>a(:,1)</code> Столбец. <code>a([1 3],[1 4])</code> Подматрица.
<code>a[-2,-3]</code>	Подматрица без второй строки и третьего столбца. В MATLABе конструкция гораздо «тяжеловеснее». Например, такая (вызывает изменение матрицы). <code>a(2,:) = [];</code> <code>a(:,3) = [];</code>
<code>t(a)</code> <code>Conj(t(a))</code>	<code>a.'</code> <code>a'</code> Транспонирование (+ сопряжение).
<code>a * b</code> <code>a %*% b</code>	<code>a .* b</code> поэлементное умножение <code>a * b</code> обычное умножение
<code>solve(a,b)</code>	<code>a \ b</code>
<code>a %*% solve(b)</code>	<code>a / b</code>
<code>a / b</code>	<code>a ./b</code> поэлементное деление
<code>a ^ k</code>	<code>a.^k</code> поэлементное возведение в степень
В «R» нет эффективного способа (только умножением матрицы на себя).	<code>a^k</code> возведение в степень
<code>rmax(x,3)</code> <code>x[x<3] <-3</code>	<code>max(x,3)</code> <code>x(x<3) = 3</code> «Обрезка». Замена всех элементов, которые

	меньше 3 на 3.
<code>solve(a)</code> <code>ginv(a)</code>	<code>inv(a)</code> Обратная матрица. <code>pinv(a)</code> Псевдобратная матрица.
<code>sqrt(sum(abs(x)^2))</code> Нет прямого аналога в R.	<code>norm(a)</code>
	<code>chol(a)</code> Нет R-аналога.
<code>qr(a)\$rank</code> Получается через QR-разложение.	<code>rank(a)</code> Ранг матрицы.
<code>sum(diag(A))</code>	<code>trace(a)</code> След матрицы.
<code>eigen(a)\$values</code> <code>eigen(a)\$vectors</code> <code>svd(a)\$d</code>	<code>eig(a)</code> <code>[v, l] = eig(a)</code> <code>svd(a)</code>
<code>tmp=eigen(A)</code> <code>w=tmp\$values</code> <code>V=tmp\$vectors</code>	<code>[V, D]=eig(A)</code> <code>w=diag(D)</code> Собственные значения и собственные векторы, записанные в матрицу.
<code>apply(a, 2, sum) # colSums(A)</code> <code>apply(a, 1, sum) # rowSums(A)</code> <code>sum(a)</code> <code>apply(a, 2, cumsum)</code>	<code>sum(a)</code> Сумма строк. <code>sum(a, 2)'</code> или <code>sum(a')</code> Сумма столбцов. <code>sum(a(:))</code> Сумма всех элементов. <code>cumsum(a)</code> Кумулятивная сумма. Аналогично с операцией максимума.
<code>cummax(x)</code> <code>cummin(x)</code>	Кумулятивный максимум и минимум. Нет простого MATLAB-аналога. Можно сделать так: <code>f = @(x,i) max(x(1:i))</code> <code>arrayfun(@(a) f(x,a), 1:length(x))</code>
<code>i <- apply(a, 1, which.max)</code>	<code>[~, i] = max(a)</code> Индексы максимальных элементов (в столбцах).
<code>rmax(b, c)</code>	<code>max(b, c)</code> Поэлементный максимум.
<code>sort(a)</code> <code>apply(a, 2, sort)</code> <code>t(apply(a, 1, sort)) # !!!</code>	<code>sort(a(:))</code> Сортировка всех элементов. <code>sort(a)</code> Сортировка по столбцам. <code>sort(a, 2)</code> Сортировка по строкам.
<code># не совсем эквивалентно</code> <code>a[order(a[, 1]),]</code> <code># Можно сортировать по</code> <code>нескольким столбцам</code> <code>order(m[, 1], m[, 2], m[, 3])</code>	<code>sortrows(a, 1)</code> Сортировка строк не имеет R-аналога. Обратите внимание на функцию <code>order</code> .
<code>apply(a, 2, mean)</code> <code>apply(a, 2, median)</code> <code>apply(a, 2, sd)</code> <code>apply(a, 2, var)</code> <code>cov(x, y)</code>	<code>mean(a)</code> <code>median(a)</code> <code>std(a)</code> <code>var(a)</code> <code>cov(x, y)</code>
<code>mean(a)</code> <code>colMeans(a) # apply(a, 2, mean)</code>	<code>mean(a(:))</code> <code>mean(a)</code>

<code>rowMeans(a) # apply(a,1,mean)</code>	<code>mean(a, 2)</code>
<code>cor(x,y)</code>	<code>a = corrccoef([1 2 3], [1 3 2])</code> <code>a(1,2)</code>
<code>a[,ncol(a):1]</code> или <code>t(apply(a,1,rev))</code> <code>a[nrow(a):1,]</code> или <code>apply(a,2,rev)</code>	<code>fliplr(a)</code> <code>flipud(a)</code> Зеркальное отражение матрицы.
<code>kronecker(matrix(1,2,3),a)</code> 2 5 2 5 2 5 1 3 1 3 1 3 2 5 2 5 2 5 1 3 1 3 1 3	<code>repmat(a,2,3)</code> «Копирование матрицы»
<code>a[lower.tri(a)] <- 0</code> <code>a[upper.tri(a)] <- 0</code>	<code>triu(a)</code> <code>tril(a)</code> Треугольные матрицы.
<code>m=dim(A)[1];</code> <code>n=dim(A)[2];</code> <code>A[(1:m-s1-1)%%m+1,(1:n-s2-1)%%n+1]</code> Нет встроенной функции.	<code>circshift(A)</code>
<code>which(a > 2)</code>	<code>find(a>2)</code>
<code>ij <- which(a != 0, arr.ind=TRUE); v <- a[ij]</code>	<code>[i j v] = find(a)</code>
<code>w = which(A > 5, arr.ind=TRUE);</code> <div> <div>row col</div> <div>[1,] 3 2</div> <div>[2,] 1 3</div> <div>[3,] 2 3</div> <div>[4,] 3 3</div> <div>[5,] 1 4</div> <div>[6,] 2 4</div> <div>[7,] 3 4</div> </div>	<code>[i,j] = find(A > 5)</code> Найти индексы элементов, удовлетворяющих заданному условию.
Размеры матриц	
<code>dim(a)</code>	<code>size(a)</code>
<code>nrow(a)</code>	<code>size(a,1)</code>
<code>ncol(a)</code>	<code>size(a,2)</code>
<code>length(dim(a))</code>	<code>ndims(a)</code>
<code>dim(a) = c(m,n)</code>	<code>a = reshape(a,m,n)</code> Изменение размеров матрицы.
<code>m <- dim(a)[1]</code> <code>n <- dim(a)[2]</code> <code>r = ((ind-1) %% m) + 1</code> <code>c = floor((ind-1) / m) + 1</code> или <code>r = row(A)[ind];</code> <code>c = col(A)[ind]</code>	<code>[r,c] = ind2sub(size(A), ind)</code> Восстановление обычной индексации по «линейной».
<code>m <- dim(a)[1]</code> <code>ind = (c-1)*m + r</code>	<code>ind = sub2ind(size(A), r, c)</code> Восстановление «линейной» индексации по обычной.

Операции со строками	
<pre>a <- 'str1' b <- 'str2' paste(a, b, sep='') [1] "str1str2"</pre>	<pre>a = 'str1'; b = 'str2'; [a b]</pre> <p>Конкатенация строк.</p>
<pre>strs = c('aa', 'bb') paste(strs, collapse='') [1] "aabb"</pre>	<pre>strs = {'aa', 'bb'}; [strs{:}]</pre> <p>Конкатенация строк в массиве ячеек / списке.</p>
<pre>str <- 'abcd' substr(str, 2, 4) [1] "bcd"</pre>	<pre>strs = 'abcd' strs[2:4]</pre> <p>Подстрока.</p>
<pre>gregexpr(p, s)[[1]]</pre>	<pre>regexp(s, p)</pre> <p>Регулярные выражения.</p>
<pre>match(c('a', 'b'), c('s', 'a', 'aa', 'cc', 'a'))</pre> <p>В R выводятся первые вхождения!</p>	<pre>[iin ipos] = ismember({'a', 'b'}, {'s', 'a', 'aa', 'cc', 'a'})</pre> <p>Проверка на вхождение элементов одного множества в другое. В системе MATLAB находятся последние вхождения.</p>
<pre>as.character(x)</pre>	<pre>num2str(x)</pre> <p>Перевод числа в строку.</p>
<pre>s = readline('Введите строку:')</pre>	<pre>s = input('Введите строку','s')</pre> <p>Ввод строки.</p>
<pre>eval(parse(text='a=pi'))</pre>	<pre>eval('a=pi')</pre> <p>Выполнение строки.</p>
Циклы	
<pre>for(i in 1:5) { print(i) }</pre>	<pre>for i=1:5 disp(i); end</pre>
<pre>c = ifelse(a>b, 1, 0)</pre>	<pre>if a>b c = 1; else c = 0; end</pre>
Списки	
<pre>x = list() x[[1]] = c(1,2) x[[2]] = 'two'</pre> <p>В «R» есть возможность обращения к элементам списка по именам.</p>	<pre>x = cell(0) x{1} = [1 2] x{2} = 'two'</pre> <p>В MATLAB есть возможность делать многомерные матрицы ячеек.</p>
Графика	
<pre>plot(x, y, type="l")</pre>	<pre>plot(x, y)</pre> <p>Обычный график.</p>
<pre>plot(x, y)</pre>	<pre>plot(x, y, 'o') scatter(x, y)</pre> <p>Скаттер-графики.</p>
<pre>plot(x1, y1, type="l") matplot(x2, y2, add=TRUE, type="l")</pre>	<pre>plot(x1, y1) hold on plot(x2, y2)</pre>
<pre>plot(x, y, type="b", col="red")</pre>	<pre>plot(x, y, 'ro-')</pre>
<pre>grid()</pre>	<pre>grid on</pre>

Использование функции <code>lines</code>	<code>hold on</code>
<code>plot(x,y, xlim=c(0,10), ylim=c(0,5) , main="title", xlab="x-axis", ylab="y-axis")</code>	<code>plot(x,y, type="l") axis([0 10 0 5]) title('title') xlab('x-axis') ylab('y-axis')</code>
<code>plot(x,y, log="y") plot(x,y, log="x") plot(x,y, log="xy")</code>	<code>semilogy(a) semilogx(a) loglog(a)</code>
<code>f <- function(x) sin(x/3) - cos(x/5) plot(f, xlim=c(0,40), type='p')</code>	<code>f = inline('sin(x/3) - cos(x/5)') ezplot(f, [0,40])</code>
<code>f <- function(x,y) x*exp(-x^2- y^2) n <- seq(-2,2, length=40) z <- outer(n,n,f) persp(n,n,z,theta=30, phi=30, expand=0.6, ticktype='detailed')</code> 	<code>n = -2:.1:2; [x,y] = meshgrid(n,n); z = x.*exp(-x.^2-y.^2); mesh(z)</code> 
<code>postscript(file="foo.eps") plot(1:3) # вывод графики dev.off()</code> <code>png(filename = "Rplot%03d.png") plot(1:3) # вывод графики dev.off()</code> В R также можно использовать следующие команды: <code>pdf(file='foo.pdf') devSVG(file='foo.svg')</code>	<code>print -depsc2 foo.eps</code> Вывод в eps-файл. <code>Print -dpng foo.png</code> Вывод в png-файл.
Другие команды	
<code>z <- lm(y~x) plot(x,y) abline(z)</code>	<code>z = polyval(polyfit(x,y,1),x) plot(x,y,'o', x,z, '-')</code> Линейная регрессия.
<code>p = coef(lm(y ~ x + I(x^2))) coef(lm(as.formula(</code>	<code>p = polyfit(x,y,2) Полиномиальная регрессия. p = polyfit(x,y,n)</code>

<code>paste('y~', paste('I(x^', 1:n, ')', sep=' ', collapse='+'))))</code>	
<code>polyroot(c(1, 2, 1))</code>	<code>roots([1 -2 1])</code> Корни полинома.
<code>uniroot(sin, c(-1, 1))</code>	<code>fzero(@(x) sin(x), 0.1)</code> Нули функции.
<code>m = optimize(sin, c(0, 2*pi))\$minimum</code>	<code>m = fminbnd(@(x) sin(x), 0, 2*pi)</code> Оптимизация.
<code>f <- function(a,b,c) {sin(a+c)+cos(b+c)} m = optimize(f, c(0, 2*pi), b=1, c=1)\$minimum</code>	<code>f = @(a,b,c) sin(a+c)+cos(b+c)</code> <code>m = fminbnd(@(a) f(a,1,1), 0, 2*pi)</code> Оптимизация функции нескольких переменных по одной переменной.
<code>integrate(sin, 0, pi)</code>	<code>quad(@(x) sin(x), 0, pi)</code> Интегрирование.
<code>fft(a, inverse=TRUE)</code>	<code>ifft(a)</code> Обратное преобразование Фурье.
нет	<code>factor(20)</code> Факторизация (разложение на множители).
<code>as.numeric(names(sort(- table(x))) [1])</code>	<code>mode(x)</code> Наиболее часто встречающееся значение.
<code>w = table(x) c = as.numeric(w) v = as.numeric(names(w))</code>	<code>v = unique(x); c = hist(x,v);</code> Перечислить все элементы мультимножества. Указать, сколько раз каждое из них входит в мультимножество.

IDE, GUI

- RStudio, <http://www.rstudio.org/>
- StatET, <http://www.walware.de/goto/statet/>
- ESS (Emacs Speaks Statistics), <http://ess.r-project.org/>
- R Commander: John Fox, “The R Commander: A Basic-Statistics Graphical Interface to R,” Journal of Statistical Software 14, no. 9 (2005):1–42.
- JGR (Java GUI for R), <http://cran.r-project.org/web/packages/JGR/index.html>
- Revolution R (коммерческая разработка): <http://www.revolutionanalytics.com/>