МАШИННОЕ ОБУЧЕНИЕ
ЛАБОРАТОРНЫЙ ПРАКТИКУМ
(предварительный вариант)

Н.Ю. Золотых, А.Н. Половинкин

9 октября 2007 г.

1. *Введение в систему R*

1.1. Основы работы с системой R

При запуске программы `RGui` появляется основное окно программы, содержащее подокно, называемое `R Console`. Это командное окно (консоль), в котором пользователь вводит команды, а система печатает результаты. В процессе работы в основном окне будет появляться другие окна, в частности, окна редактора скриптов, графические окна, в которых изображаются графические результаты выполнения команд, и др. Команды вводятся пользователем (в командном окне) после приглашения, которое имеет вид

```
>
```

Команда поступает системе на обработку, как только пользователь нажал Enter. В одной строке можно набрать несколько команд, разделяя их символом `;` (точка с запятой). Если клавиша Enter нажата, а команда еще не завершена, то приглашение приобретает вид

```
+
```

В дальнейшем тексте любая строка, начинающаяся с `>`, означает клавиатурный ввод: вам необходимо набрать то, что напечатано после `>`, но без знака `>`. Символ `+` продолжения команды в настоящем тексте мы будем всегда опускать.

Часть строки, начинающаяся с символа `#` и заканчивающаяся концом строки (там, где вы нажали Enter), — это *комментарии*. Они игнорируются системой.

Чтобы получить справку о функции, имя которой вам известно, нужно набрать

```
> help(имя-функции)
```

Имя интересующей вас функции может стоять в кавычках (одинарных или двойных) или без кавычек. Например,

```
> help("lm")
> help('lm')
> help(lm)
```

Если справка по указанной функции имеется, то откроется новое окно с содержимым этой справки. Справка доступна также через пункт меню Help.

R поддерживает работу с историей команд. В командной строке с помощью клавиш «стрелка вверх» и «стрелка вниз» можно прокручивать (в обоих направлениях) список всех введенных ранее команд. Как только нужная команда найдена, ее можно отредактировать, используя клавиши Del, «стрелка влево», «стрелка вправо» и алфавитно-цифровые клавиши.

Если команды записаны в некотором файле, например, `myprog.r`, то запустить из на выполнение можно с помощью команды

```
> source("myprog.r")
```

Эта возможность доступна также через меню FILE.

Чтобы перенаправить весь вывод в файл, скажем, `myprog.out`, воспользуйтесь командой

```
> sink("myprog.out")
```

Восстановить вывод на консоль можно командой

```
> sink()
```

Для выхода из R нужно набрать

```
> q()
```

или воспользоваться меню.

1.2. Числа

Основным типом данных в системе R является число с плавающей точкой двойной точности. Для отделения дробной части от целой используется точка. Показатель отделяется от мантиссы символом `e` или `E`. Например, `31415`, `+3.1415e4`, `.31415e+5`, `3141500e-2` — разные записи одного и того же числа 31415.

Над числами можно выполнять обычные арифметические операции: `+`, `-`, `*`, `/`, `^` (степень), `%/%` (целочисленное деление), `%%` (остаток от деления). Операции имеют обычный приоритет. В выражениях используются круглые скобки. Набранное в командной строке арифметическое выражение после нажатия на ENTER вычисляется и результат сразу же отображается:

```
>  2*(7-9)^8+6/3
[1] 514
```

Ответ — это `514`. Смысл `[1]` будет ясен из дальнейшего.

Элементарные функции, разумеется, реализованы. Вот (неполный) список: абсолютное значение `abs`, квадратный корень `sqrt`, экспонента e^x `exp`, натуральный логарифм $\ln x$ `log`, десятичный логарифм $\log_{10} x$ `log10`, двоичный логарифм $\log_2 x$ `log2`, тригонометрические функции `sin`, `cos`, `tan`, обратные тригонометрические функции `asin`, `acos`, `atan`. Аргумент функций записывается в круглых скобках. Например,

```
> exp(1)
[1] 2.718282
> (sqrt(5) + 1)/2
[1]  1.618034
> sin(pi/2)
[1] 1
```

где `pi` — это константа π. Если у функции несколько аргументов, то они отделяются знаком `,` (запятая). Например, $atan2(y, x)$ возвращает угол между осью Ox и вектором (x, y).

Доступны следующие функции округления. Функция `ceiling(x)` находит минимальное целое, не меньшее `x` (округление в сторону $+\infty$). Функция `floor(x)` возвращает максимальное целое, не превосходящее `x` (округление в сторону $-\infty$). Функция `trunc(x)` отбрасывает дробную часть (округление в сторону 0). Функция `round(x)` округляет к ближайшему целому. Функция `round(x, dig)` с дополнительным параметром `dig` указывает, что необходимо осуществить округление до заданного знака после запятой.

Особую роль в R играют следующие объекты: бесконечность `Inf`, «не-число» `NaN`, «отсутствующее значение» `NA`. Бесконечность `Inf` (может иметь также отрицательный знак) получается при переполнениях и операциях вида $1/0$ и т. п. «Не-число» `NaN` возникает как результат операций вида `0/0`, `Inf-Inf` и т. п. Если в результате вычислений получается `NaN`, то R выдает предупреждение. Значение `NA` (not available) означает, что значение не доступно. Пользователь сам может в выражениях использовать `Inf`, `NaN` и `NA`.

Любое числовое значение можно приписать какой-либо переменной. Эту переменную можно использовать в последующих командах[1]. Например,

```
> x <- sqrt(5)
> phi <- (x + 1)/2
```

Здесь `x` и `phi` — это имена переменных, а знак `<-`, означающий присваивание[2], состоит из двух символов: `<` и `-`. Заметим, что присваиваемые значения не будут напечатаны в командном окне. Чтобы посмотреть на присвоенное значение, нужно еще раз набрать имя переменной. Например,

```
> phi
[1]  1.618034
```

Имена переменных (идентификаторы) в R состоят из строчных и заглавных букв[3], цифр и знаков `.` (точка) и `_` (подчеркивание). Идентификатор не может начинаться с цифры, и если он начинается с точки, то цифра не может быть его вторым символом[4]. R чувствителен к регистру, т. е. строчные и заглавные буквы в нем различаются. Например, имена `phi` и `Phi` относятся к разным переменным.

Список всех имеющихся на данный момент переменных можно получить с помощью функции `ls()`. Удалить переменную можно с помощью функции

[1] Никогда ничего не присваивайте переменной `pi`. Иначе у вас не будет доступа к константе π.

[2] Вместо `<-` можно использовать `=`. Кроме того, возможно присваивание в виде `(x + 1)/2 -> phi`.

[3] Можно даже использовать кириллицу, если ваша операционная система ее поддерживает, но для переносимости программ лучше этого не делать.

[4] Мы были удивлены, когда обнаружили, что в R возможны переменные, имена которых состоят только из точек — по-видимому, из любого их количества, но не трех. Лексема `...` в R используется для других целей. Разумеется, мы не рекомендуем использовать переменные с такими именами.

`rm()`, перечислив в списке аргументов имена переменных, подлежащих удалению.

1.3. Числовые векторы

Векторы в R формируются функцией `c()`. Аргументы этой функции суть компоненты вектора. Например, команда

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

формирует вектор-строку `(1, 2, 3, 4, 5)`. Разумеется, векторы можно присваивать переменным:

```
> a = c(1, 2, 3, 4, 5)
```

Аргументами функции `c()` могут являться векторы. В этом случае как результат получаем конкатенацию этих векторов. Скалярные значения (т. е. числа) воспринимаются R как векторы длины 1. Таким образом, аргументами функции `c()` могут быть как векторы, так и скаляры. Например,

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

Вектор, состоящий из последовательных чисел, можно получить с помощью команды `<начальное-значение>:<конечное-значение>`. Например, `1:5`. На эту команду похожа функция `seq`, которая генерирует отрезки арифметической прогрессии. Можно задать начальное значение, конечное значение и шаг:

```
> seq(0, 1, by = 0.1)
[1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

В другом варианте задается начальное значение, конечное значение и количество точек. В результате будет сгенерирован вектор, состоящий из заданного числа компонент, равномерно распределенных на заданном отрезке:

```
> seq(0, 1, len = 11)
[1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

Функция `rep(v, k)` создает вектор, состоящий из `k` копий вектора `v`. Например,

```
> rep(c(1, 2), 3)
[1] 1 2 1 2 1 2
```

Функция `runif(n)` создает вектор указанной длины со случайными элементами, равномерно распределенными на отрезке $[0, 1]$ Функция `rnorm(n)` создает вектор указанной длины `n` со случайными элементами, распределенными по нормальному закону с математическим ожиданием 0 и дисперсией 1.

Элементы вектора могут иметь значения `Inf`, `NaN`, `NA`. Например,

```
> age = c(23, NA, NA, 18, 19)
```

Над векторами можно выполнять арифметические операции и элементарные функции. Бинарная операция над двумя векторами одинаковой длины производится почленно над каждой парой элементов, и результатом является вектор, состоящий из результатов этого действия. В случае, когда размерность векторов не совпадает, производится приведение длины более короткого вектора к длине длинного. Приведение выполняется следующим образом: вектор приобретает длину, такую же, как и у более длинного, и содержимое этих новых элементов берётся из содержимого старых элементов вектора, повторяемых циклически.

```
> c(1, 2, 3, 4) + c(1, 2)
[1] 2 4 4 6
> c(1, 2, 3, 4) - c(1, 2)
[1] 0 0 2 2
> c(1, 2, 3, 4) * c(1, 2)
[1] 1 4 3 8
> c(1, 2, 3, 4)/c(1, 2)
[1] 1 1 3 2
> c(1, 2, 3, 4)^c(1, 2)
[1]  1  4  3 16
> c(1, 2, 3, 4)/c(1, 0)
[1]   1 Inf   3 Inf
> 2*c(1, 2, 3, 4, 5)
[1]  2  4  6  8 10
```

Если длина короткого вектора не кратна длине более длинного, то выдается предупреждение, но результат вычисляется. Например,

```
> c(3, 1, 4, 1, 5, 9, 2) + c(9, 5)
[1] 12 6 13 6 14 14 11
```

Элементарные математические функции применяются к каждой компоненте вектора. Например,

```
> x = c(0, pi/2, pi)
> sin(x)
[1] 0.000000e+00 1.000000e+00 1.224606e-16
```

Доступ к элементам вектора осуществляется оператором `[i]`. Например, `u[5]` — это 5-й элемент вектора `u`. Нумерация элементов начинается с 1. Выражения вида `u[i]` могут встречаться и в левой части от знака присваивания. При этом если вектор имеет длину не меньше `i`, то `u[i]` просто примет новое значение. В противном случае вектор u увеличит свою длину до i, компонента `u[i]` примет новое значение, а остальные новые компоненты примут значения `NA`.

```
> u <- 1
```

```
> u[5] <- 5
> u
[1]  1 NA NA NA  5
```

Некоторые полезные функции:
`sum(v)` — сумма элементов вектора `v`;
`prod(v)` — произведение компонент вектора `v`;
`max(v)` — максимальный элемент;
`min(v)` — минимальный элемент;
`length(v)` — длина вектора;
`mean(v)` — среднее значение элементов вектора `v` (оценка математического ожидания);
`var(v)` — вариация элементов вектора `v` (оценка дисперсии);
`sort(v)` — возвращает вектор той же длины, что и `v`, с элементами, отсортированными в порядке возрастания.

1.4. Логические векторы

R умеет работать с логическими векторами (и, следовательно, с логическими скалярми), элементы которого могут иметь значения `TRUE` и `FALSE`, а также значение `NA`. Логические векторы получаются в результате сравнений и применения к логическим векторам логических функций. Операции сравнения — это `<`, `<=`, `>`, `>=`, `==` (для равенства) и `!=`(для неравенства). Логические функции — это `&` (и), `|` (или), `!` (не). Операнды могут иметь разную длину. Сравнения и логические функции выполняются покомпонентно и, если требуется, с циклическим сдвигом, как и в случае арифметических операций. Например,

```
> young <- (age >= 2) & (age <= 50)
```

создает логический вектор `young` той же длины, что и `age`, с компонентами, равными `TRUE`, где условие выполнено, и `FALSE`, где условие не выполнено.

```
> age = c(1, 2, NA, Inf, NaN, 18, 19)
> young <- (age >= 2) & (age <= 50)
> young
[1] FALSE  TRUE    NA FALSE    NA  TRUE  TRUE
```

Логические векторы могут использоваться в обычной арифметике. При этом `TRUE` интерпретируется как `1`, а `FALSE` как `0`.

Функция `is.na(a)` возвращает логический вектор той же длины, что и `a`, с компонентами, равными `FALSE`, где соответствующие значения `a` равны `NaN` или `NA`, и `TRUE` в остальных случаях. Функция `is.nan(a)` возвращает логический вектор той же длины, что и `a`, с компонентами, равными `FALSE`, где соответствующие значения `a` равны `NaN`, и `TRUE` в остальных случаях. Например,

```
> a = c(0, 1, Inf, NaN, NA)
```

```
> is.na(a)
[1] FALSE FALSE FALSE  TRUE  TRUE
> is.nan(a)
[1] FALSE FALSE FALSE  TRUE FALSE
```

1.5. Символьные векторы

Чтобы определить в R строку символов, достаточно заключить ее в двойные или одинарные кавычки. При этом `\n` означает переход на новую строку, `\t` — табуляцию, `\b` — возврат на символ назад, `\\` — символ `\`, `\"` — символ `"`, `\'` — символ `'`. Из отдельных строк можно компоновать векторы. Например,

```
> label = c("Petal width", "Petal length")
> label
[1] "Petal width"  "Petal length"
```

Конкатенацию строк осуществляет функция `paste`. В простейшем варианте, например, происходит склейка указанных строк со вставкой в качестве разделителя пробела. Например,

```
> paste("Sepal width", "Sepal length", "Petal width",
                                                "Petal length")
[1] "Sepal width Sepal length Petal width Petal length"
```

Если аргументы функции `paste` — массивы, то склеиваются соответствующие компоненты. При этом, если длины массивов различны, происходит циклическая подстановка меньшего из них. Например,

```
> paste(c("Sepal", "Petal"), c("width", "length"))
[1] "Sepal width"  "Petal length"
> paste(c("Sepal", "Petal"),
                      c("width", "width", "length", "length"))
[1] "Sepal width"  "Petal width"  "Sepal length"
                                                 "Petal length"
```

Если один из векторов — числовой, то он автоматически будет конвертирован в символьный:

```
> paste("x", 1:5)
> [1] "x 1" "x 2" "x 3" "x 4" "x 5"
```

Можно заменить разделитель на другой:

```
> paste("x", 1:5, sep = "")
> [1] "x1" "x2" "x3" "x4" "x5"
```

1.6. Строковые наименования элементов вектора

Элементам вектора (числового, логического, символьного) можно задавать имена. Например,

```
> fruit <- c(5, 10, 1, 20)
> names(fruit) <- c("orange", "banana", "apple", "peach")
> fruit
orange banana  apple  peach
     5     10      1     20
```

Мы создали вектор длины 4, компоненты которого имеют указанные названия. Теперь обращаться к элементам вектора можно как по индексу, так и по имени:

```
> fruit[1] <- 6
> fruit["peach"] <- 60
> fruit
orange banana  apple  peach
     6     10      1     60
```

1.7. Векторы в качестве индексов

Пусть `v` — некоторый вектор (числовой, логический, символьный). Напомним, что обращение к конеретному элементу этого вектора осуществляется с помощью команды индексирования `v[i]`. Оказывается, что в качестве индекса `i` здесь может выступать не скаляр, а вектор. В этом случае генерируется срез исходного вектора `v`, состоящий из тех его компонент, которые удовлетворяют определенным условиям.

Как будет интерпретирован вектор, рассматриваемый в качестве индекса, зависит от его типа. Возможны следующие случаи.

- `i` — это логический вектор. В этом случае длины векторов i и v должны совпадать. В результате `v[i]` — это вектор, сформированный из тех и только тех компонент вектора `v`, для которых соответствующее значение в `i` есть `TRUE`.

- `i` — это вектор с положительными целыми значениями. В этом случае компоненты вектора `i` интерпретируются как обычные индексы и `v[i]` — это вектор, сформированный из соответствующих компонент вектора `v` в той последовательности, в которой они указаны в `i`.

- `i` — это вектор с отрицательными целыми значениями. Абсолютные значения вектора `i` интерпретируются как номера компонент, которые исключаются из `v`.

- `i` — это символьный вектор. Его значения интерпретируются как имена компонент вектора `v`. В результате `v[i]` — это вектор, сформированный

из соответствующих компонент вектора v в той же последовательности, в которой они указаны в i.

Рассмотрим примеры:

```
> y <- x[!is.na(x)]
```

формирует вектор y только из инициализированных некоторым значением (не NaN и NA) компонент вектора x.

Команда

```
> x[is.na(x)] <- 0
```

заменяет отсутствующие значения нулями.

Выражение

```
> z <- (x + 1)[(!is.na(x)) & x > 0]
```

создает вектор z и размещает в нем элементы вектора x + 1, удовлетворяющие приведенным условиям.

Команда

```
> x[1:10]
```

формирует вектор из первых 10 компонент вектора x (предполагается, что длина вектора x больше 10).

Команда

```
> x <- x[-(1:5)]
```

удаляет из x первые 5 компонент.

Приведем еще один пример. Пусть f — вектор, содержащий протабулированные значения некоторой функции в точках, хранящихся в x. Требуется найти корни функции. Это можно сделать следующим образом:

```
> n <- length(x)
> w <- 1:(n - 1)
> x(f[w] == 0 | (f[w]*f[w + 1] < 0))
```

1.8. Графики функций

Для изображения графиков функций в R есть функция plot(x, y). Здесь x — вектор значений абсцисс и y — вектор значений ординат. Если указан только один аргумент — plot(y), то предполагается, что x=1:n, где n — длина вектора y. Например, команды

```
> x = seq(-pi, pi, len = 30)
> y = sin(x)
> plot(x, y)
```

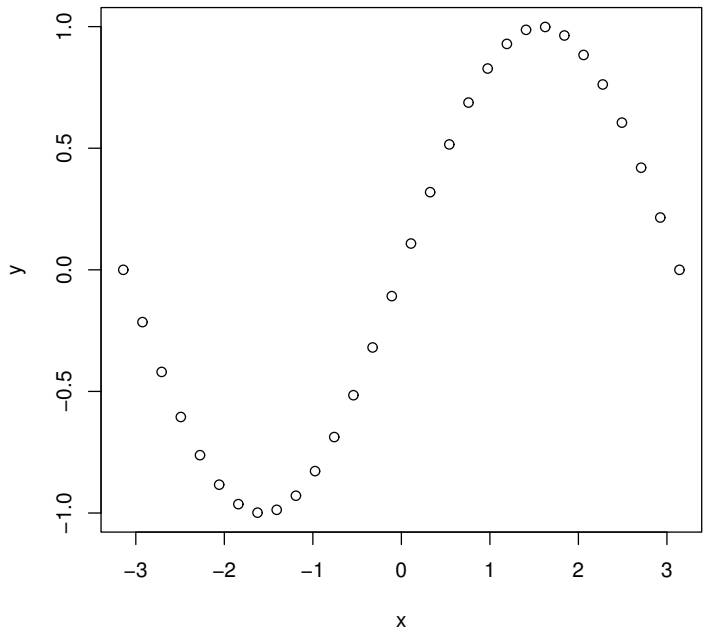

Рис. 1.1. «Точечный» график синуса

рисуют 30 точек, лежащих на синусоиде `y = sin(x)`; см. рис. 1.1. Графики появятся в отдельном окне (или в нескольких отдельных окнах), называемых графическими устройствами.

Функция `plot` допускает множество дополнительных параметров. Вот некоторые из них.

- `log = "x"` или `log = "y"` или `log = "xy"` делает соответственно масштаб по оси абсцисс, ординат или по обеим осям логарифмическим;
- `type = тип-графика` позволяет указать тип выводимого графика; некоторые значения параметра — `"p"` (точки, по умолчанию), `"l"` (линии), `"b"` (линии и точки), `"o"` (линии и точки перекрываются), `"n"` (ничего не рисуется);
- `xlab = название-оси-абсцисс` и `ylab = название-оси-ординат` позволяет указать подписи к оси абсцис и ординат соответственно;
- `main = основная-надпись` и `sub = дополнительная-надпись` создают надписи сверху и снизу графика;
- `col = цвет` задает цвет графика; возможны значения `"blue"`, `"red"`, `"green"`, `"cyan"`, `"magenta"`, `"yellow"`, `"black"` и многие другие. Можно

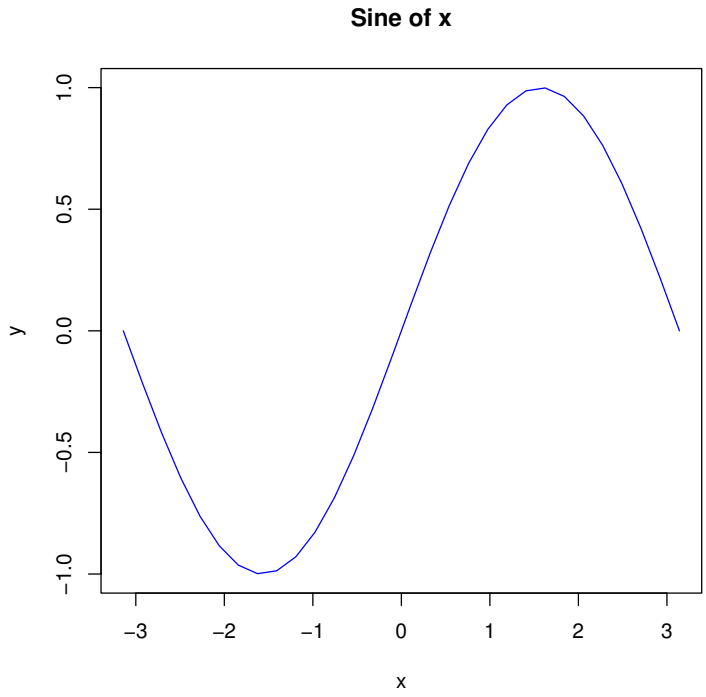

Рис. 1.2. График синуса

задать rgb-вектор функцией `rgb(r, g, b)`. Параметры этой функции — значения от `0` до `1`.

- `lty = стиль-линии` определяет стиль линии; возможны значения `"blank"` (нет линии), `"solid"`, `"dashed"`, `"dotted"`, `"dotdash"`, `"longdash"`, `"twodash"`.

Рассмотрим следующий пример:

```
> plot(x, y, type = "l", col = "blue", main = "Sine of x")
```

Следующие функции (называемые низкоуровневыми) никогда не стирают имеющегося изображения. Им могут быть переданы также дополнительные параметры, аналогичные соответствующим параметрам функции `plot`.

`points(x, y)` рисует дополнительные точки, `lines(x, y)` рисует ломаную линию, соединяющую указанные точки. Этим функциям могут быть переданы дополнительные аргументы, например, `type = тип-графика` (по умолчанию он равен `"p"` для `points` и `"l"` для `lines`).

`text(x, y, текст)` добавляет текст. По умолчанию он центрируется вокруг точки `x`, `y`. Указанные параметры могут быть векторами. В этом случае будет размещено несколько надписей. Можно указать шрифт `font = шрифт` и др.

`abline(k, b)` рисует прямую `y=kx + b`. Можно указать цвет и стиль линиии и т. д.

`abline(h = y)` `abline(v = x)` — для отрисовки горизонтальных и вертикальных прямых. Указывается соответственно координата `y` и `x`.

`polygon(x, y)` рисует многоугольник с вершинами `x`, `y`. Можно указать цвет заполнения `col = цвет` и цвет границы `border = цвет`.

`legend(x, y, легенда)` добавляет легенду в указанную точку.

Рассмотрим пример

```
> t = seq(0, 10, by = 0.1)
> y1 = 1.03*exp(-0.25*t)*sin(0.97*t)
> y2 = 1.07*exp(-0.35*t)*sin(0.94*t)
> y3 = 1.15*exp(-0.5*t)*sin(0.87*t)
> y4 = 0.45*(exp(-0.38*t) - exp(-2.62*t))
> y5 = 0.22*(exp(-0.21*t) - exp(-4.80*t))
> plot(t, y1, type = "n", main = "Oscillation")
> colors = c("blue", "red", "green", "cyan", "magenta")
> lines(t, y1, col = colors[1])
> lines(t, y2, col = colors[2])
> lines(t, y3, col = colors[3])
> lines(t, y4, col = colors[4])
> lines(t, y5, col = colors[5])
> legends = paste("y", 1:5, sep = "")
> legend(6, 0.6, legends, lty = "solid", col = colors)
> abline(h = 0) # Горизонтальная ось
> abline(v = 0) # Вертикальная ось
```

Результат приведен на рис. 1.3

1.9. Специальная графика

Функция `barplot(x)` рисует столбцовую диаграмму. При этом каждой компоненте вектора `x` будет соответствовать столбец. Если компоненты вектора `x` имеют имена, то они будут использоваться в качестве подписей к столбцам диаграммы. Задать подписи можно с помощью дополнительного параметра `names.arg = names`. Определить цвет диаграммы можно с помощью дополнительного параметра `col = colors`. Здесь `colors` — либо число или строка, обозначающие цвет, которым будет нарисована диаграмма, либо вектор, каждая компонента которого определяет цвет каждого столбца.

Рассмотрим пример

```
> area <- c(17075200, 9976140, 9639810, 9598077,
                                        8511965, 7741220)
> names(area) <- c("Russia", "Canada", "USA", "China",
                                   "Brazil", "Australia")
> barplot(area, main = "Area", col = c("red", "green3",
                  "brown3", "yellow", "blue", "green"))
```

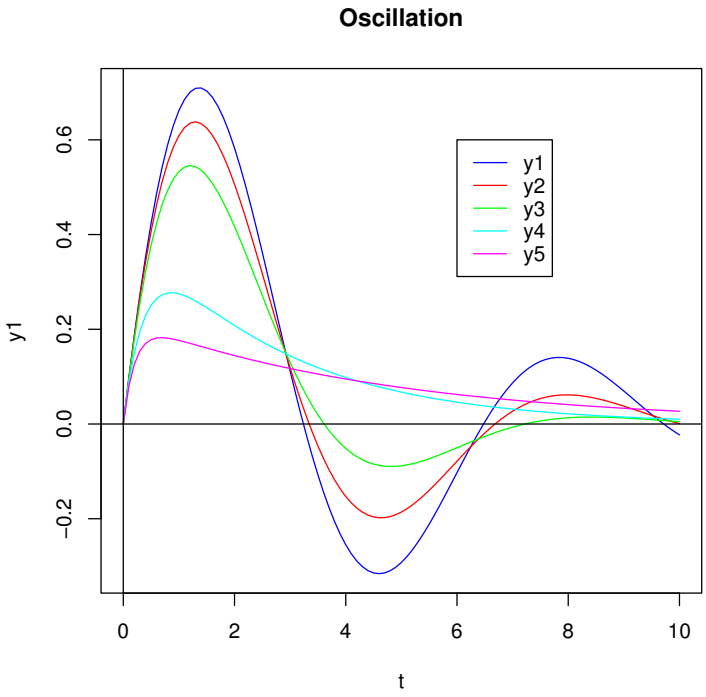

Рис. 1.3. Несколько графиков в одном графическом окне

Результат представлен на рис. 1.4.

Теперь рассмотрим случай, когда аргумент `A` функции `barplot(A)` — матрица. В этом случае каждому ее столбцу соответствует столбец диаграммы, составленный из маленьких столбиков, поставленных друг на друга. Высоты столбиков определяются соответствующими значениями, хранящимися в `A`. Если указано значение параметра `beside = TRUE` (по умолчанию `beside = FALSE`), то столбики будут поставлены не друг на друга, а рядом. Значение `legend.text = TRUE` включает отображение легенды. В качестве подписей к легенде берутся имена строк матрицы `A`. Также эти подписи можно указать явно, задав в качестве значения параметра `legend.text = TRUE` символьный массив.

Рассмотрим пример

```
> data = c(
      12416505,  13130000,
      12626921,  13060000,
       8814860,  10170000,
       3995077,   4218000,
       3779044,   4156000,
       1552008,   1746000,
       1566253,   1655000)
```

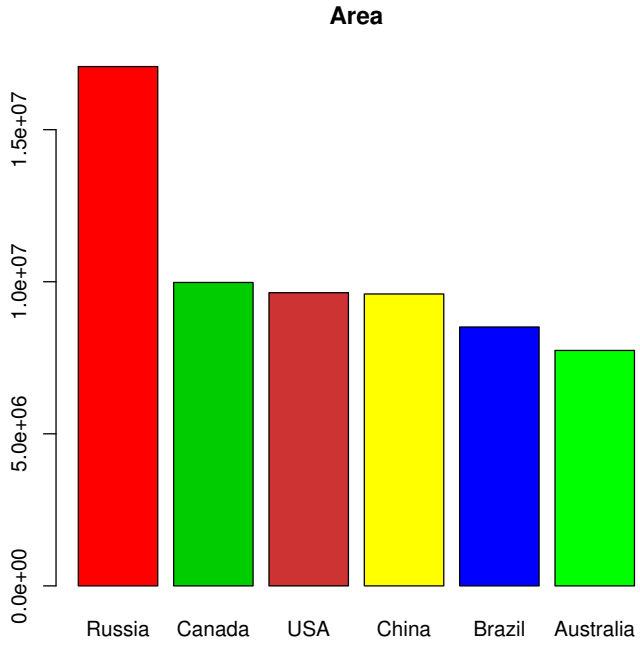

Рис. 1.4. Столбцовая диаграмма

```
> gdp <- matrix(data, nrow = 2)
> colnames(gdp) <- c(
      "USA", "EU", "China", "Japan",
      "India", "Russia", "Brazil")
> rownames(gdp) <- c(2005, 2006)
> barplot(gdp, beside = TRUE, col = 1:2,
      main = "GDP", legend.text = TRUE)
```

Результат представлен на рис. 1.5.

Функция `pie(x)` строит круговую диаграмму. Использование этой функции аналогично использованию функции `barplot`, однако допустим только вариант, когда `x` — вектор (не матрица). Рассмотрим пример:

```
> big.islands <- islands[islands < islands["Australia"]
                         & islands >= islands["Sumatra"]]
> big.islands
> pie(sort(big.islands, decreasing = TRUE),
                              col = 1:length(big.islands))
```

Результат представлен на рис. 1.6.

Функция *hist* строит гистограмму. Количество отрезков, на которое разбивается интервал, выбирается автоматически, но можно его указать явно,

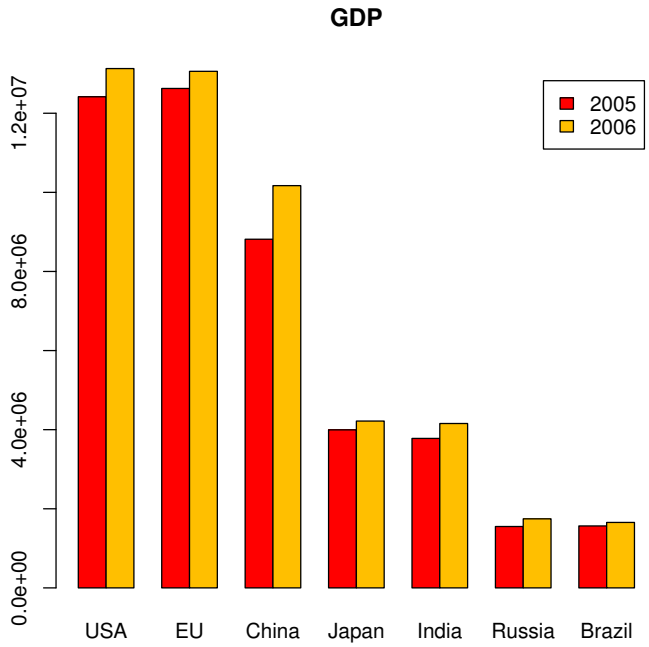

Рис. 1.5. Столбцовая диаграмма

определив значение параметра `breaks`.

```
> set.seed(0)
> x <- rnorm(200)
> hist(x, breaks = 14, col = "cyan")
```

Результат приведен на рис. 1.7.

1.10. Списки

Списки в R — это коллекции объектов, доступ к котором можно проиводить по номеру или имени. Список может содержать объекты (компоненты) разных типов, что отличает списки от векторов. Компонентами списка могут быть в том числе векторы и другие списки.

Функция

```
> list(объект1, объект2, ...)
```

создает список, содержащий указанные объекты. Например,

```
>  Euler = list("Leonhard Euler", "Mathematician", NA)
```

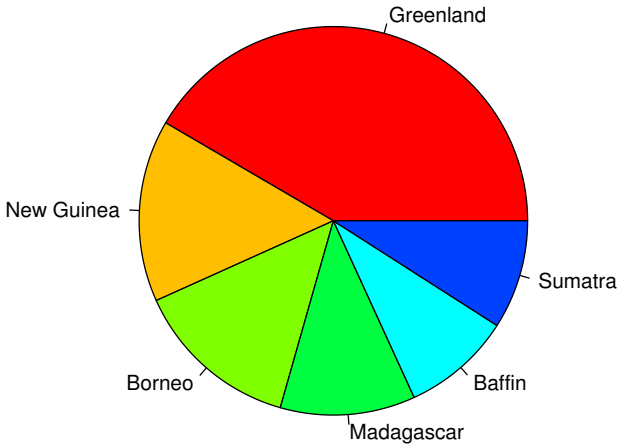

Рис. 1.6. Круговая диаграмма

```
>  Euler
[[1]]
[1] "Leonhard Euler"

[[2]]
[1] "Mathematician"

[[3]]
[1] NA
```

К компонентам можно обращаться по номеру. Номер указывается в двойных квадратных скобках после имени списка. Например,

```
> Euler[[2]]
[1] "Mathematician"
> Euler[[3]] <- 1707
```

Можно создавать новые компоненты:

```
> Euler[[4]] <- TRUE
> Euler
[[1]]
```

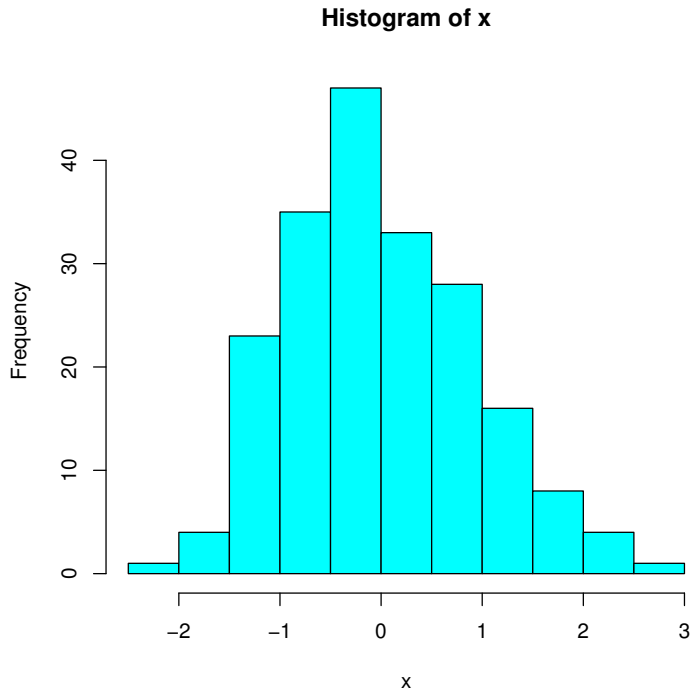

Рис. 1.7. Гистограмма

```
[1] "Leonard Euler"

[[2]]
[1] "Mathematician"

[[3]]
[1] 1746

[[4]]
[1] TRUE
```

Функция `list()` без аргументов создает пустой список.

Можно задать имена компонент так же, как мы задавали имена для компонент вектора:

```
> names(Euler) <- c('name', 'occupation', 'year', 'married')
> Euler
@name
[1] "Leonhard Euler"

@occupation
```

```
[1] "Mathematician"

@year
[1] 1707

@married
[1] TRUE
```

Теперь к компонентам вектора можно обращаться по имени. Для этого есть две возможности. Имя можно указывать либо после знака `@`, который приписывается к имени списка, например, `Euler@occupation`, либо в кавычках и двойных квадратных скобках, например, `Euler[["occupation"]]`. Последняя возможность удобна тем, что вместо самого имени компоненты может стоять имя переменной (разумеется, уже без кавычек) или даже целое выражение, значение которого есть имя компоненты.

Имена компонент списка можно указать сразу при его создании:

```
>  Euler = list(name = "Leonhard Euler", occupation = "Mathematician")
```

Одинарные квадратные скобки создают срезы списка. Например, `lst[1:4]` — список, состоящий из первых четырех компонент списка `lst`, `lst[-(1:4)]` — список, полученный исключением первых четырех компонент из списка `lst`, `lst[c("name", "occupation")]` — список, состоящий из указанных компонент вектора и т. д. Правила использования индексов аналогичны соответствующим правилам для векторов.

Важно понимать различие между правилами использования одинарных и двойных квадратных скобок. Например, `lst[[1]]` — это *список*, состоящий из первой компоненты списка `lst`, тогда как `lst[1]` — это сама первая компонента. Отметим также, что в двойных квадратных скобках в качестве индексов не могут использоваться векторы.

Списки — это рекурсивный тип данных, т. е. компоненты списка могут сами быть списками. Например,

```
> Pushkin <- list(name = "Александр Сергеевич Пушкин", year = 1799)
> Gogol <- list(name = "Николай Васильевич Гоголь", year = 1809)
> lst <- list(Pushkin, Gogol)
> lst[[1]]@name
[1] "Александр Сергеевич Пушкин"
> length(lst)
[1] 2
```

Функция `c` осуществляет конкатенацию двух или более списков.

```
> clst <- c(Pushkin, Gogol)
> clst
@name
[1] "Александр Сергеевич Пушкин"

@year
```

```
[1] 1799

@name
[1] "Николай Васильевич Гоголь"

@year
[1] 1799

> length(clst)
[1] 4
> clst@name
[1] "Александр Сергеевич Пушкин"
> clst@year
[1] 1799
> clst[[3]]
[1] "Николай Васильевич Гоголь"
> clst[[4]]
[1] 1809
```

1.11. Матрицы

Числовую матрицу можно создать из числового вектора с помощью функции `matrix`. Нужно указать число строк `ncol = m` и/или число столбцов `ncol = m`. В результате элементы из вектора будут записаны в матрицу указанных размеров. Например,

```
> matrix(1:6, nrow = 2, ncol = 3)
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Длина вектора должна быть кратна произведению требуемого числа строк на требуемое число столбцов:

```
> matrix(1, nrow = 2, ncol = 2)
     [,1] [,2]
[1,]    1    1
[2,]    1    1

> matrix(1:2, nrow = 3, ncol = 2)
     [,1] [,2]
[1,]    1    2
[2,]    2    1
[3,]    1    2
```

Если указывается только одна из размерностей (например, только число столбцов), то она должна быть кратна длине вектора. Вторая размерность будет равна отношению длины вектора к первой размерности.

Как мы видим элементы записываются в матрицу по столбцам. Чтобы записать их по строкам, нужно задать значение дополнительного параметра `byrow = TRUE`:

```
> matrix(1:6, nrow = 2, byrow=TRUE)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

Функции `nrow(A)` и `ncol(B)` возвращают число строк и столбцов матрицы A соответственно.

Доступ к элементам матрицы происходит по индексу. `A[i, j]` ссылается на элемент `i`-й строки и `j`-го столбца матрицы `A`. На месте индексов `i` и `j` могут стоять векторы. Интерпретация такая же, как и у векторов. Если, например, `i` и `j` — это векторы с целыми положительными элементами, то `A[i, j]` есть подматрица матрицы `A`, образованная элементами, стоящими на пересечении строк с номерами из `i` и столбцов с номерами из `j`. Если, `i` и `j` — это векторы с целыми отрицательными элементами, то `A[i, j]` есть подматрица, полученная из `A` вычеркиванием соответствующих строк и столбцов. Векторы `i` и `j` могут быть логическими или символьными. В последнем случае строкам и столбцам матрицы должны быть присвоены имена (см. ниже).

`A[i, ]` эквивалетно `A[i, 1:ncol(A)]`, а `A[, j]` эквивалетно `A[1:nrow(A), j]`.

Возможен доступ к элементам матрицы с помощью одного индекса. Например, `U[5]` означает `5`-й элемент матрицы `U`, если считать, что элементы перенумерованы по столбцам. Если `i` — вектор, то `A[i]` — означает выборку соответствующих элементов и т. д.

Можно задать имена столбцам и строкам матрицы

```
> A <- matrix(c(23, 31, 58, 16), nrow = 2)
> rownames(A) <- c('petal', 'sepal')
> colnames(A) <- c('length', 'width')
```

После этого доступ к строкам и столбцам может происходить по имени. Например, `A['petal', 'length]`.

Функция

```
> A <- cbind(A, B)
```

создает матрицу, приписывая к `A` справа `B` (для этого число строк у `A` и `B` должно совпадать). Функция

```
> A <- rbind(A, B)
```

создает матрицу, приписывая к `A` снизу `B` (для этого число столбцов у `A` и `B` должно совпадать). Заметим, что в списках аргументов функций `cbind` и `rbind` можно указать более двух матриц.

Арифметические операции над матрицами осуществляются покомпонентно, поэтому, например, чтобы сложить две матрицы, они должны иметь одинаковые размеры:

```
> A <- matrix(1:6, nrow = 2)
> B <- matrix(-(1:6), nrow = 2)
> A + B
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
```

Впрочем, можно осуществлять смешанные операции, когда один из операндов — матрица, а другой — вектор (в частности, скаляр). В этом случае матрица рассматривается как вектор, составленный из ее элементов, записанных по столбцам, и действуют те же правила, что и для арифметических операций над векторами:

```
> (1:2)*A
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    4    8   12
> (1:3) + A
     [,1] [,2] [,3]
[1,]    2    6    7
[2,]    4    5    9
> A^2
     [,1] [,2] [,3]
[1,]    1    9   25
[2,]    4   16   36
```

Элементарные математические функции также применяются к элементам матрицы покомпонентно.

Функция `outer(x, y, op)` применяет операцию `op` к каждой паре элементов векторов `x` и `y`. На выходе получаем матрицу, составленную из результатов выполнения этой операции. Число строк матрицы есть длина вектора `x`, а число столбцов — длина вектора `y`. Например,

```
> x <- 1:3
> x
[1] 1 2 3
> y <- -2:2
> y
[1] -2 -1  0  1  2
> outer(x, y, "+")
     [,1] [,2] [,3] [,4] [,5]
[1,]   -1    0    1    2    3
[2,]    0    1    2    3    4
[3,]    1    2    3    4    5
> outer(x, y, "*")
     [,1] [,2] [,3] [,4] [,5]
[1,]   -2   -1    0    1    2
[2,]   -4   -2    0    2    4
[3,]   -6   -3    0    3    6
```

Вместо `outer(x, y, "*")` (*внешнее* произведение векторов) можно использовать `x %o% y`.

Транспонирование матрицы осуществляет функция `t(A)`, а матричное произведение — операция `%*%`:

```
> A <- matrix(1:6, nrow = 2)
> B <- t(A)
> A%*%B
     [,1] [,2]
[1,]   35   44
[2,]   44   56
> B%*%A
     [,1] [,2] [,3]
[1,]    5   11   17
[2,]   11   25   39
[3,]   17   39   61
```

Для решения системы линейных уравнений `Ax = b` с квадратной невырожденной матрицей `A` есть функция `solve(A, b)`:

```
> A <- matrix(1:4, ncol = 2)
> A
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> b = c(5, 8)
> solve(A, b)
[1] 2 1
```

`det(A)` находит определитель, а `solve(A)` — обратную матрицу:

```
> det(A)
> [1] -2
> solve(A)
     [,1] [,2]
[1,]   -2  1.5
[2,]    1 -0.5
```

Среди других процедур, относящихся к линейной алгебре, в R есть методы нахождения разложения Холецкого (функция `chol`), QR-разложения (`chol`), сингулярного разложения (`svd`), нахождение собственных векторов и собственных чисел (`eigen`) и др.

1.12. Многомерные массивы

Матрицы — это частный случай *многомерных массивов*. Матрицы имеют две размерности. В общем случае массивы могут иметь больше размерностей.

Работа с многомерными массивами в R во многом аналогична. Основной способ их создания — функция `array`. Указываются элементы массива и все его размерности. Например, создадим массив размера $2 \times 3 \times 4$:

```
> A = array(1:24, c(2, 3, 4))
> A
, , 1

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

, , 2

     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12

, , 3

     [,1] [,2] [,3]
[1,]   13   15   17
[2,]   14   16   18

, , 4

     [,1] [,2] [,3]
[1,]   19   21   23
[2,]   20   22   24
```

Можно задать имена размерностям:

```
> dim1 <- c('Income', 'Expence')
> dim2 <- c('John', 'George', 'Paul')
> dim3 <- c('Winter','Spring','Summer','Autumn')
> dimnames(A) <- list(dim1, dim2, dim3)
```

и после этого обращаться к элементу по имени его строки, столбца, слоя и т. д.

Заметим, что подобным образом происходит работа с массивами, состоящими из символьных строк, логическими массивами и массивами, полученными на основе списков.

1.13. Факторы

Фактор — это векторный объект, кодирующий категориальные данные (классы). Факторы создаются с помощью функции `factor`. Предположим, что

у нас имеется 3 класса `Yes`, `No`, `Perhaps`. И некоторая выборка из 6 объектов, каждый из которых принадлежит одному из этих классов. Тогда

```
> v = c("Yes", "No", "Yes", "Perhaps", "No", "Perhaps")
> f = factor(v)
> f
[1] Yes     No      Yes     Perhaps No      Perhaps
Levels: No Perhaps Yes
> length(f)
[1] 6
```

1.14. Фреймы данных

Фреймы данных (data frames) — один из самых важных типов данных в R, позволяющий объединять данные разных типов вместе. Можно считать, что фрейм данных — эта двумерная таблица, в которой (в отличие от числовых матриц), разные столбцы могут содержать данные разных типов (но все данные в одном столбце имеют один тип). Например, такая таблица может содержать результаты эксперимента.

Создать фрейм данных можно с помощью функции `data.frame`:

```
> frm <- data.frame(data1, data2, ...)
```

Здесь многоточие означает, что список данных может содержать произвольное число элементов. В качестве данных (`data1`, `data2`, ...) могут выступать векторы (числовые, символьные или логические), факторы, матрицы (числовые, символьные или логические), списки или другие фреймы. При этом все векторы должны иметь одинаковую длину, а матрицы и фреймы — одинаковое (такое же) число строк. Могут также встречаться векторы, длина которых меньше, но в этом случае эта длина должна являться делителем максимальной встречающейся длины. То же требование предъявляется к компонентам списков. Функция `data.frame` просто собирает все данные вместе. Символьные векторы конвертируются в факторы. Остальные данные собираются во фрейм такими, какие они есть.

Рассмотрим пример:

```
> Y <- matrix(c(1950, 1980, 1973, 1995, 1978, 2001,
                                    1983, 2005), nrow = 4)
> rownames(Y) <- c("Иванов", "Петров", "Сидоров", "Смит")
> colnames(Y) <- c("born", "entrance")
> Y
        born entrance
Иванов  1950     1978
Петров  1980     2001
Сидоров 1973     1983
Смит    1995     2005
> m <- c(TRUE, TRUE, TRUE, FALSE)
```

```
> c <- c(2, 1, 3, 0)
> P <- data.frame(Y, m, c)
> P
         born entrance     m c
Иванов   1950     1978  TRUE 2
Петров   1980     2001  TRUE 1
Сидоров  1973     1983  TRUE 3
Смит     1995     2005 FALSE 0
```

Мы видим, что в результате создан фрейм данных с 4 строками и 4 столбцами. Строкам и столбцам присвоены имена. В качестве имен строк и имен первых двух столбцов использованы имена строк и столбцов матрицы `Y`. Если бы строки и столбцы этой матрицы не были именованы, то имена для строк и столбцов результирующего фрейма назначаются автоматически. В качестве имен двух последних столбцов фрейма взяты имена переменных, из которых эти столбцы были получены.

Имена можно изменить:

```
> rownames(P) <- 1:4
> colnames(P)[3] <- "married"
> colnames(P)[4] <- "nchildren"
> P
  born entrance married nchildren
1 1950     1978    TRUE         2
2 1980     2001    TRUE         1
3 1973     1983    TRUE         3
4 1995     2005   FALSE         0
```

Если среди аргументов функции `data.frame` есть список, то каждая его компонента превращается в столбец фрейма. Например,

```
> lst = list(1:4, TRUE, c("A", "B"))
> data.frame(lst)
  X1.4 TRUE. c..A....B..
1    1  TRUE           A
2    2  TRUE           B
3    3  TRUE           A
4    4  TRUE           B
```

Доступ к элементам фрейма осуществляется двумя способами: в «матричном» или «списковом» стилях. В первом случае указываются номера или имена строки и столбца. Во втором фрейм рассматривается как список, компонентами которого являются столбцы фрейма, и чтобы обратиться к его элементу нужно указать имя или номер этой компоненты (т. е. имя или номер столбца), а затем имя или номер строки. Рассмотрим примеры:

```
> P[1, "born"]
[1] 1950
> P[4, 3]
```

```
[1] FALSE
> P[, "nchildren"]
[1] 2 1 3 0
> P[2:3, ]
    X1   X2 married nchildren
2 1980 2001    TRUE         1
3 1973 1983    TRUE         3
> P$born
[1] 1950 1980 1973 1995
> P$nchildren[3]
[1] 3
> P[["born"]][2]
[1] 1980
> P[[1]][2]
[1] 1980
```

1.15. Графики функций двух переменных

```
hot.colors <- function(n)
{
    m <- trunc(n/3)
    m3 <- n - 2*m
    pas <- 0.75

    r <- c(seq(0, pas, len = m), rep(1, n - m))
    g <- c(rep(0, m), seq(0, pas, len = m), rep(1, n - 2*m))
    b <- c(rep(0, 2*m), seq(0, pas, len = n - 2*m))
    rgb(r, g, b)
}
n = 300
x <- y <- seq(-1, 1, len = n)
X <- matrix(rep(x, n), nrow = n)
Y <- t(X)
Z <- (X^2 + Y^2)^3 - 4*X^2*Y^2
contour(x, y, Z, levels = c(-0.1, -0.05, -0.01, -0.001, 0, 0.1, 0.2), col = "blue")
# filled.contour(x, y, Z, levels = c(-0.1, -0.05, -0.01, -0.001, 0, 0.1, 0.2),  color.palett
# persp(x, y, Z)


# rainbow cm.colors topo.colors terrain.colors heat.colors


n = 44
x <- y <- seq(-pi, pi, len = n)
r <- (outer(x^2, y^2, "+"))
z <- exp(-sqrt(r))*cos(r)
```

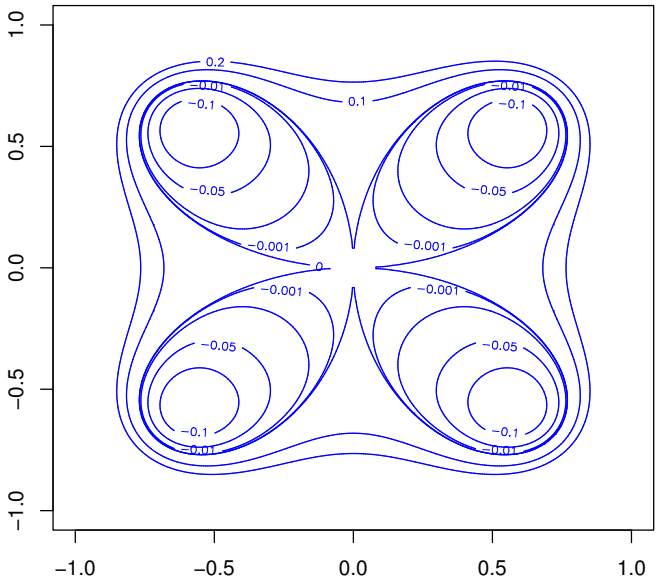

Рис. 1.8. Линии уровня функции $y = (x^2 + y^2)^3 - 4x^2y^2$.

```
persp(x, y, z, theta = 30, phi = 30, expand = 0.5, col = "red", ltheta = 120, shade = 0.4, t
```

1.16. Ввод и вывод

1.17. Управляющие конструкции

Каждая команда или выражение в языке R возвращает некоторый результат. Даже присваивание — это выражение, результат которого — присваиваемое значение. Выражение может быть частью другого выражения. В частности, возможны множественные присваивания, например,

```
> x <- y <- z <- 0
```

Выражения можно объединять в *блоки*. Блок — это несколько выражений, заключенных в фигурные скобки. Сами выражения, как обычно, разделяются точкой с запятой или символом перехода на новую строку. Результатом выполнения всего блока будет результат последнего из выражений, составляющих блок:

```
> {
```

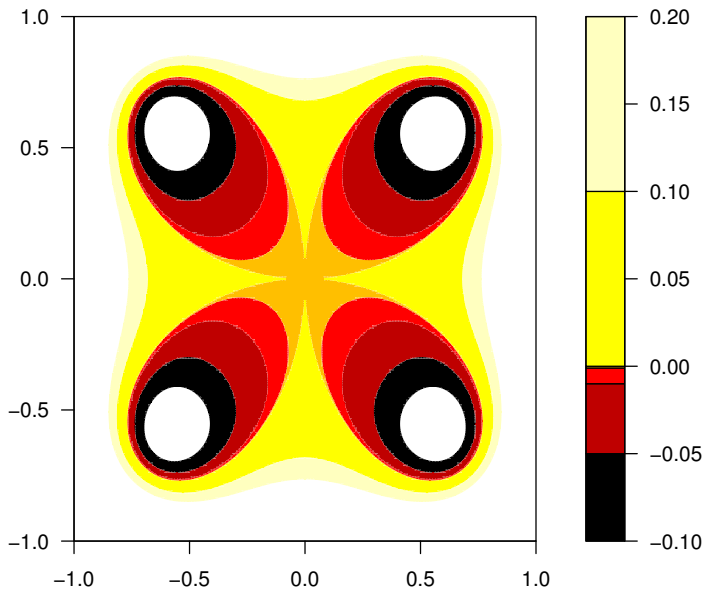

Рис. 1.9. Линии уровня с закрашенными промежутками

```
      x <- 7; y <- 8
      x + y
  }
[1] 15
```

Проверка условий осуществляется так же, как в языке C:

```
> if (условие) выражение1 else выражение2
```

где `условие` — это логическое выражение, результат которого есть скалярное логическое значение. Часто условия содержат логические операции `&&`, `||`. Тогда как `&`, `|` — покомпонентные операции, применяемые к векторам, `&&`, `||` применяютя только к скалярным значениям и вычисляют свой второй операнд лишь при необходимости.

Циклы с управляющей переменной реализуются с помощью конструкции

```
> for (переменная in выражение1) выражение2
```

Результатом `выражения1` должен быть вектор, при этом `переменная` на каждой итерации цикла принимает значение очередного элемента этого вектора. Количество итераций равно количеству элементов в векторе.

```
> s = 0
```

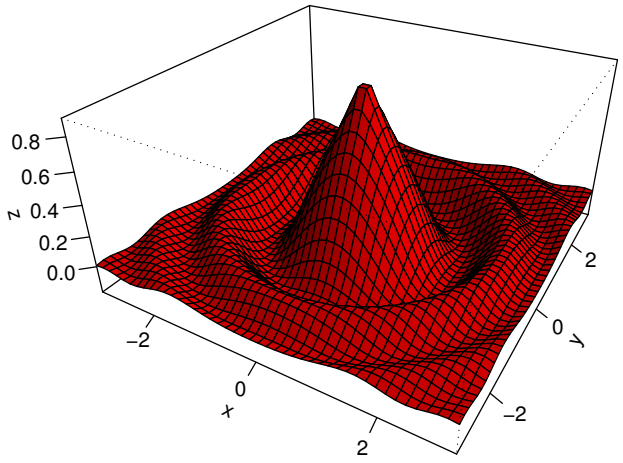

Рис. 1.10. График функции $z = \cos(x^2 + y^2)e^{\sqrt{(x^2+y^2)}}$

```
> for (i in 1:20) s = s + i
> s
[1] 210
```

Заметим, что тот же результат быстрее можно получить с помощью команды `sum(1:20)`. Заметим, что циклы встречаются в R намного реже, чем в «традиционных» языках. Гораздо проще и эффективнее с точки зрения времени выполнения использовать функции, работающие сразу со всем объектом, чем с его частями.

Циклы с предусловием реализуются с помощью конструкции

```
> while (условие) выражение
```

Преждевременный выход из цикла осуществляется командой `break`. Для прерывания текущей итерации и перехода к следующей служит команда `next`. Эти команды можно также использовать и с конструкцией `for`.

«Бесконечный» цикл реализует команда

```
> repeat выражение
```

Внутри этой конструкции можно использовать команды `break` и `next`.

1.18. Функции пользователя

Формат определения функции следующий:

```
> имя <- function(арг1, арг2, ...) выражение
```

Здесь многоточие означает, что список *формальных аргументов* (*формальных параметров*) `арг1`, `арг2`, ... может иметь произвольную длину (в том числе, быть пустым). Как правило, `выражение` (*тело* функции) представляет собой блок. Значение, возвращаемое функцией, это значение этого выражения. Досрочное возвращение из функции осуществляет команда `return(выражение1)`. В этом случае функция возвращает значение указанного выражения.

Обращение к функции имеет вид `имя(выражение1, выражение2, ...)`. Здесь значения выражений `выражение1`, `выражение2`, ... являются фактическими аргментами, которые подставляются вместо соответствующих формальных аргументов.

Действуя обычным образом, меняя значения формальных аргументов внутри функции, вы не меняете соответствующих фактических аргументов. Единственный способ сделать это — использовать в теле функии «сильное» присваивание: `arg1 <-- выражение`

Рассмотрим несколько простых примеров

Если фактические аргументы заданы в «именованном» виде `имя = выражение` (многочисленные примеры этого мы видели раньше), то они могут быть перечислены в произвольном порядке. Более того, список фактических аргументов может начинаться с аргументов, представленных в обычной позиционной форме, после чего могут идти аргументы в именованной форме.

Например, имеется функция `fun`, заданная следующим образом:

```
> fun <- function(arg1, arg2, arg3, arg4)
  {
      # тело функции опущено
  }
```

Тогда `fun` может быть вызвана многочисленными способами, например:

```
> ans <- fun(d, f, 20, TRUE)
> ans <- fun(d, f, arg4 = TRUE, arg3 = 20)
> ans <- fun(arg1 = d, arg3 = 20, arg2 = f, arg4 = TRUE)
```

Все эти способы эквивалентны.

Во многих случаях при определении функции некоторым аргументам могут быть присвоены значения по умолчанию. Тогда при вызове функции эти аргменты могут быть опущены. Предположим, что функция `fun` определена как

```
> fun <- function(arg1, arg2, arg3 = 20, arg4 = TRUE)
  {
      # тело функции опущено
  }
```

Тогда обращение к этой функции вида

```
> fun(d, f)
```

Эквивалентно трем, приведенным выше. Следующее обращение

```
> fun(d, f, arg4 = FALSE)
```

меняет одно из значений по умолчанию.

Заметим, что в значениях по умолчанию можно использовать любые выражения, в том числе включающие другие аргменты функции.

Все встречающиеся в теле функции символы делятся на три группы: это формальные аргументы, локальные переменные и свободные переменные.

Формальные аргументы (формальные параметры) — это аргументы, перечисленные в заголовке функции (в круглых скобках после ключевого слова `function`.

Локальные переменные — это переменные, не являющиеся формальными аргументами, значения которых определяются во время выполнения функции.

Переменные, не являющиеся формальными аргументами и локальными переменными, являются *свободными переменными.*

Например, в функции

```
> f <- function(x)
  {
        y <- 2*x
        print(x)
        print(y)
        print(z)
  }
```

x — формальный аргумент, y — локальная переменная, z — свободная переменная.

Область видимости формальных аргументов и локальных переменных — только сама эта функция. Это означает, что изменения этих переменных внутри функции никак не отражаются на переменных с такими же именами во внешней функции. Область видимости свободных переменных распространяется до внешней функции, в которой они были определены. Изменение таких переменных в теле функции влияет также на соответствующие переменные в этой внешней функии.

1.19. boxplot и мозаичные диаграммы

```
> colnames(chickwts)
[1] "weight" "feed"
> nrow(chickwts)
[1] 71
>boxplot(weight ~ feed, data = chickwts, col = "salmon",
```

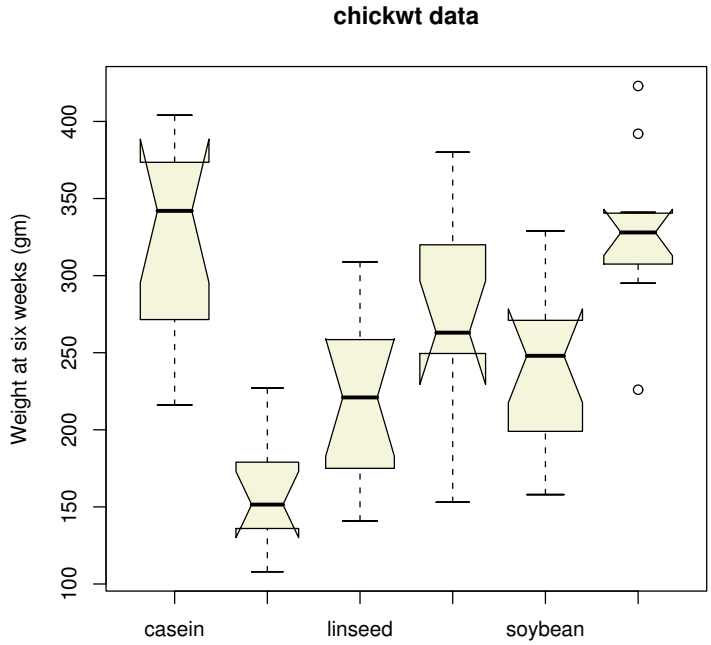

Рис. 1.11. boxplot

```
    varwidth = TRUE, notch = TRUE, main = "chickwt data",
    ylab = "Weight at six weeks (gm)")
```

Графический результат см. на рис. 1.11.

Таблица `Titanic` из библиотеки `datasets` содержит информацию о пассажирах Титаника. Таблица имеет 4 размерности:

`"Class"`: `Titanic["1st", , ,]`, `Titanic["2nd", , ,]`, `Titanic["3rd", , ,]`, `Titanic["Crew", , ,]`,

`"Sex"`: `Titanic[, "Male", ,]`, `Titanic[, "Female", ,]`,

`"Age"`: `Titanic[, , "Child",]`, `Titanic[, , "Adult",]`,

`"Survived"`: `Titanic[, , , "Yes"]`, `Titanic[, , , "No"]`.

Элементы таблицы — количество человек из данной группы. Проверим нуль-гипотезу о том, что общее количество выживших пассажиров и членов экипажа не зависит от класса билета. Построим таблицу сопряженности признаков и мозаичную диаграмму:

```
> titanic <- apply(Titanic, c(1, 4), sum)
> titanic
      Survived
Class   No Yes
  1st  122 203
```

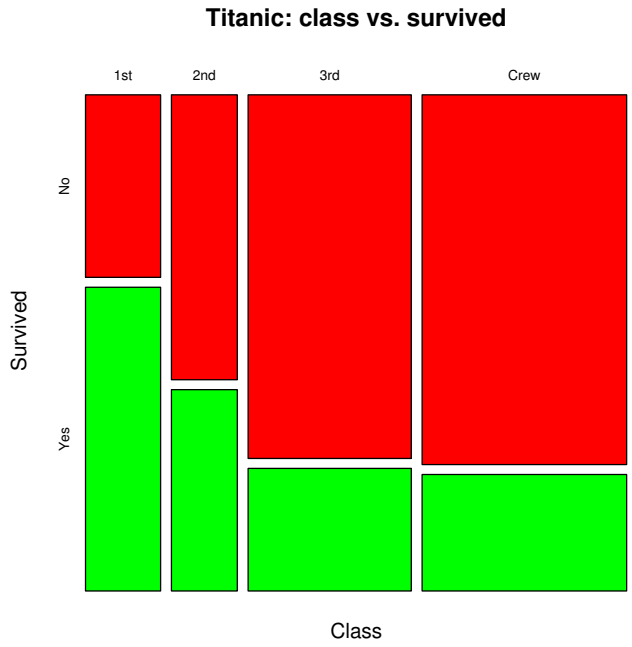

Рис. 1.12. Мозаичная диаграмма. Доля выживших на Титанике пассажиров и членов команды в зависимости от класса их билета.

```
    2nd  167 118
    3rd  528 178
    Crew 673 212
> mosaicplot(titanic, col = c("red", "green"), main = "Titanic: class vs survived")
```

Графический результат см. на рис. 1.12.

1.19.1. Задания к лабораторной работе

1) Сгенерируйте вектор длины `N=1000`, элементами которого являются реализации нормально распределенной случайной величины с математическим ожиданием, равным `1`, и стандартным отклонением, равным `0.3`. Подсчитайте статистическое мат. ожидание и стандарную ошибку, *не* используя встроенные функции и проверьте правильность результата. Подсчитайте `.95,.99`-квантили. Исследуйте отклонение статистического мат. ожидания от `1` при росте `N (N=1000,2000,4000,8000)`.

2) Создайте фрейм данных из `N=20` записей со следующими полями: `Nrow` — номер записи, `Name` — имя пользователя, `BirthYear` — год рождения, `EmployYear` — год приема на работу, `Salary` — зарплата, где `Nrow` изменя-

ется от 1 до N, Name задается произвольно, BithYear распределен равномерно на отрезке [1960,1985], EmployYear распределен равномерно на отрезке [BirthYear+18,2006], Salary для работников младше 1975 г.р. определяется по формуле Salary=(ln(2007-EmployYear)+1)*8000, для остальных Salary=(log2(2007-EmployYear)+1)*8000. Подсчитайте число сотрудников с зарплатой, большей 15000. Добавьте в таблицу поле, соответствующее суммарному подоходному налогу (ставка 13%), выплаченному сотрудником за время работы в организации, если его зарплата за каждый код начислялась согласно формулам для Salary, где вместо 2007 следует последовательно подставить каждый год работы сотрудника в организации.

3) Напишите функцию, которая принимает на вход числовой вектор x и число разбиений интервала k (по умолчанию равное числу элементов вектора, разделенному на 10) и выполняет следующее: находит минимальное и максимальное значение элементов вектора xmin и xmax, разделяет полученный отрезок [xmin;xmax] на k равных интервалов и подсчитывает число элементов вектора, принадлежащих каждому интервалу. Далее должен строиться график, где по оси абсцисс — середины интервалов, по оси ординат — число элементов вектора, принадлежащих интервалу, разделенное на общее число точек. Проведите эксперимент на данной функции, где x — вектор длины 5000, сгенерированный из экспоненциально распределенной случайной величины,k=500. Приближение какого графика мы получаем в итоге при большом числе точек и числе разбиений?

4) Спроектируйте и реализуйте метод наименьших квадратов.