Введение в R

Дружков П.Н., Золотых Н.Ю., Половинкин А.Н.

20 сентября 2013 г.

Содержание

1 Общие сведения

Язык программирования R является диалектом (реализацией) языка S, который был разработан как эффективное и удобное средство для работы с

данными, моделирования и визуализации. R развивается в рамках open-source-проекта и доступен для различных платформ: Windows, MacOS и UNIX/UNIX-подобных систем (включая FreeBSD и Linux).

Полезные ссылки:

- Официальный сайт проекта: http://www.r-project.org
- Краткий справочник часто используемых функций:
  http://cran.r-project.org/doc/contrib/Short-refcard.pdf (англ.),
  http://aakinshin.blogspot.ru/2013/06/r-functions.html (рус.)
- Популярная свободная среда разработки: http://www.rstudio.com

2 Основы работы в системе R

Основным способом работы пользователя в системе R является использование интерактивного режима работы в консоли интерпретатора, либо написание программ-скриптов для их дальнейшего исполнения в системе. Как правило, скрипты сохраняются в файлах, имеющих расширение .R. Вызов скрипта на исполнение из консоли интерпретатора языка осуществляется функцией `source("path/to/file.R")`. При этом можно указывать относительные пути, отправной точкой которых является рабочая директория, узнать которую можно с помощью функции `getwd()`, а изменить с помощью `setwd("new/working/directory")`.

Все вычисления производятся в некотором окружении (environment), которое определяет связь между именами и значениями: между именем переменной и ее значением, между именем функции и ее реализацией. Данное связывание производится путем объявления и инициализации переменных с помощью оператора присваивания (`=` или `<-`)[1]. Дополнительного указания типа переменной при ее создании не требуется.

```
> x = 7
> x
[1] 7
> y <- 1.5
> y
[1] 1.5
```

Следует отметить, что в приведенном выше примере (как и во всех примерах в данном пособии) приведен результат интерактивного выполнения кода в консоли интерпретатора, где каждая новая команда пишется после приглашения (по умолчанию символа «>»), при этом в тексте скритпов данные символы должны быть опущены. Результат работы кода одинаков в этих двух режимах за исключением вывода результата на экран, для чего в скриптах необходимо использовать функции `print(x)` или `sprintf(fmt, ...)`. Функция `sprintf` служит для форматированного вывода и имеет такой же формат, как и аналогичная функция в языке C. Для перенаправления вывода в файл используется функция `sink(file)`. Восстановление вывода в консоль производится вызовом данной функции без аргументов.

Узнать все доступные в текущем окружении имена можно с помощью функции `ls()`. Удалить некоторое имя `x` можно с помощью функции `rm(x)`,

[1] Объявление функций рассматривается далее в данном пособии.

полностью очистить текущее окружение можно следующим способом `rm(list = ls())`.

Каждая команда или выражение в языке R возвращает некоторый результат. Выражение может быть частью другого выражения. В частности, возможны множественные присваивания. 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.

Для получения справки по той или иной функции можно выполнить команду `help("function_name")`.

В последующих разделах данного пособия приводится краткая информация по существующим типам (структурам данных) языка R, основами программирования на данном языке, работе с пакетами расширений и графикой.

3 Базовые типы

3.1 Вектор

Векторы[2] являются основополагающей струтурой данных языка R, представляющей собой некоторое количество однотипных элементов непрерывно расположенных в памяти. В R есть 6 базовых типов векторов (или, что тоже самое, их элементов): логический (logical), целочисленный (integer), вещественный (double), комплексный (complex), символьный (character) и двоичные данные (raw). Любая константа и переменная одного из перечисленных выше типов также является вектором длины 1. Для определения длины вектора служит функция `length(x)`.

```
> length(7.8)
[1] 1
> length("text")
[1] 1
```

К числовым векторам можно применять арифметические операции (`+`, `-`, `*`, `/`, `^`), которые будут выполнены поэлементно. Если длины векторов различны, то вектор меньшей длины будет повторен нужное количество раз и обрезан до нужного размера. К целочисленным векторам аналогично применимы операции получения целой части `%/%` и остатка `%%` от деления. Аналогичным образом векторы можно сравнивать с помощью операций `==`, `!=`, `<`, `<=`, `>`, `>=`, результатом применения которых являются логические векторы. К логическим векторам, в свою очередь, можно применять логические операции `&` (конъюнкция), `|` (дизъюнкция), `!` (отрицание). Элементы векторов кроме обычных значений, обусловленных их типом, могут принимать специальные значения: `NA` (недоступное, пропущенное значение), `NaN` (не число), `Inf` (бесконечность), `-Inf` (минус бесконечность).

[2] В терминологии языка программирования R различают атомарные (atomic) и общие (generic) векторы. Последние могут содержать элементы различных типов. Так, списки являются векторами, но не атомарными. В данном пособии для облегчения терминологии, говоря о векторах, мы будем подразумевать атомарные.

3.1.1 Функции создания векторов

Приведем описание[3] некоторых базовых функций для создания векторов.

- `integer(n)`, `double(n)`, `numeric(n)`, `complex(n)`, `logical(n)`, `character(n)`, `raw(n)`. Создают векторы длины n соответствующего типа, инициализированные значениями по умолчанию. `numeric(n)` создает вектор чисел с плавающей запятой, т.е. типа `double`.

```
> integer(10)
[1] 0 0 0 0 0 0 0 0 0 0
> logical(5)
[1] FALSE FALSE FALSE FALSE FALSE
```

- `c(...)`. Объединяет (конкатенирует) несколько векторов в один.

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

- Оператор `:`. Создает вектор, состоящий из последовательных целых чисел, границы интервала при этом указываются следующий образом: <начальное_значение>:<конечное_значение>.

```
> 1:5
[1] 1 2 3 4 5
```

- `seq(from, to, by = d)`, `seq(from, to, length = n)` – Разбивает отрезок `[from, to]` на части с шагом `d`, либо на `(n - 1)` равных частей.

```
> 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
> seq(0, 1, len = 5)
[1] 0.00 0.25 0.50 0.75 1.00
```

- `rep(x, times)`. Конкатенирует `times` копий вектора `x`.

```
> rep(c(TRUE, FALSE), 3)
[1]  TRUE FALSE  TRUE FALSE  TRUE FALSE
```

- Оператор `[`. Создает вектор, состоящий из заданных элементов исходного вектора. Таким образом, `a[idx]` представляет собой новый вектор, содежащий элементы вектора `a`, указанные в `idx`. При этом вектор `idx` может состоять из положительных целых чисел и содержать в себе индексы элементов в `a` (индексация начинается с 1), которые нужно включить в новый вектор, либо состоять из отрицательных целых чисел и содержать в себе индексы элементов в `a`, которые нужно исключить из исходного вектора, либо быть логической маской.

```
> (10:20)[c(1, 3, 5)]
[1] 10 12 14
> (10:20)[-c(1, 3, 5)]
[1] 11 13 15 16 17 18 19 20
```

[3] Здесь и далее приводится не полное описание агрументов функций. Подробное описание может быть найдено в документации.

- `sample(x, n, replace = FALSE, prob = NULL)`. Создает вектор, являющийся случайной выборкой `n` значений из вектора `x` с возвращением или без в зависимости от значения параметра `replace`. Распределение вероятностей можно задать с помощью параметра `prob`, по умолчанию все значения равновероятны.

```
> sample(1:10, 5, replace = FALSE)
[1] 5 8 2 1 9
> sample(1:10, 5, replace = TRUE)
[1] 10  2  6  3 10
```

3.1.2 Математические функции

Приведем описание некоторых математических функций, применяемых к векторам.

- `sin(x)`, `cos(x)`, `tan(x)`, `asin(x)`, `acos(x)`, `atan(x)`, `atan2(x)`. Тригонометрические функции.
- `exp(x)`, `log(x)`, `log10(x)`, `log(x, base)`. Экспонента и логарифм.
- `max(x)`, `min(x)`. Поиск максимального/минимального значения.
- `range(x)`. Поиск минимума и максимума, т.е. `c(min(x), max(x))`.
- `sum(x)`. Сумма элементов вектора.
- `prod(x)`. Произведение элементов вектора.
- `mean(x)`. Среднее арифметическое элеметов вектора.
- `median(x)`. Медиана (средний по величине элемент) вектора.
- `quantile(x, probs = seq(0, 1, 0.25))`. Подсчет выборочных квантилей указанных уровней.
- `var(x)`. Несмещенная оценка дисперсии.
- `sd(x)`. Оценка стандартного (среднеквадратичного) отклонения.

3.1.3 Вероятностные распределения

Приведем описание некоторых функций для генерации выборок реализаций случайных величин, распределенных по определенным законам. Все перечисленные ниже функции принимают в качестве своего первого параметра объем требуемой выборки (размер вектора, получаемого на выходе функции).

- `rnorm(n, mean = 0, sd = 1)`. Выборка из нормального распределения.
- `runif(n, min = 0, max = 1)`. Выборка из равномерного распределения.
- `rexp(n, rate = 1)`. Выборка из экспоненциального распределения.
- `rt(n, df)`. Выборка из распределения Стьюдента.
- `rf(n, df1, df2)`. Выборка из распределения Фишера.

- `rchisq(n, df)`. Выборка из распределения Пирсона.
- `rbinom(n, size, prob)`. Выборка из биномиального распределения.
- `rgeom(n, prob)`. Выборка из геометрического распределения.
- `rhyper(nn, m, n, k)`. Выборка из гипергеометрического распределения.
- `rlogis(n, location = 0, scale = 1)`. Выборка из логистического распределения.
- `rlnorm(n, meanlog = 0, sdlog = 1)`. Выборка из логнормального распределения.
- `rnbinom(n, size, prob)`. Выборка из отрицательного биномиального распределения.

3.1.4 Другие функции

Приведем описание еще нескольких полезных функций для работы с векторами.

- `sort(x, decreasing = FALSE)`. Создает вектор, состоящий из элементов вектора `x`, отсортированных в порядке убывания (`decreasing = TRUE`) или возрастания (`decreasing = FALSE`).
- `order(x, decreasing = FALSE)`. Создает вектор-перестановку элементов вектора `x` для его упорядочивания по возрастанию или убыванию. Сортировка значений вектора `x` может быть осуществлена следующим образом: `x[order(x)]`.
- `round(x)`, `trunc(x)`, `floor(x)`, `ceiling(x)`. Функции округления векторов с числами с плавающей запятой.
- `is.integer(x)`, `is.double(x)`, `is.numeric(x)`, `is.complex(x)`, `is.logical(x)`, `is.character(x)`, `is.raw(x)`. Функции для проверки типа вектора, возвращают логическое значение.
- `as.integer(x)`, `as.double(x)`, `as.numeric(x)`, `as.complex(x)`, `as.logical(x)`, `as.character(x)`, `as.raw(x)`. Функции конвертирования типов векторов.
- `is.na(x)`, `is.nan(x)`, `is.infinite(x)`. Функции для проверки являются ли элементы вектора равными `NA`, `NaN`, `Inf`/`-Inf`.
- `paste(..., sep = "", collapse = NULL)`. Функция для конкатенирования векторов, сконвертированных в символьные. Данная функция конвертирует все, переданные в нее векторы в символьные, после чего поэлементно конкатенирует их, используя разделитель `sep`. Если указан параметр `collapse`, то все элементы полученного вектора также объединяются в одну строку, используя разделитель `collapse`.

```
> paste(c("a", "b", "c"), 1:3, sep = "-")
[1] "a-1" "b-2" "c-3"
> paste(c("a", "b", "c"), 1:3, sep = "-", collapse = "/")
[1] "a-1/b-2/c-3"
> paste(c("a", "b", "c"), collapse = "")
[1] "abc"
```

3.2 Матрица

В то время как векторы представляют одномерную последовательность данных базовых типов, зачастую удобнее оперировать многомерными данными. Так матрицы в языке R представляют собой двумерные векторы. Для преобразования вектора в матрицу можно присвоить вектору атрибут размерности с помощью функции `dim(x)`. Данная функция принимает в качестве аргумента вектор, матрицу или массив, возвращая вектор размерностей. Вектор может иметь либо пустой вектор размерностей, либо вектор размерностей длины один, содержа в себе длину вектора. Матрица должна иметь вектор размерностей длины 2, первая компонента которого обозначает количество строк, вторая – столбцов. Данные в матрице хранятся по столбцам.

```
> x = 1:16
> dim(x)
NULL
> dim(x) = c(16)
> x
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
> dim(x)
[1] 16
> dim(x) = c(2, 8)
> x
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    3    5    7    9   11   13   15
[2,]    2    4    6    8   10   12   14   16
```

Другим способом создания матриц является использование функции `matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE)`. Параметр `data` принимает вектор, элементами которого будет проинициализированна созданная матрица. Параметры `nrow` и `ncol` определяют количество строк и столбцов соответственно, `byrow` определяет будет ли заполнена матрица элементами из `data` по строкам или по столбцам.

```
> x = 1:16
> y = matrix(data = x, nrow = 2, ncol = 8)
> x
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
> y
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    1    3    5    7    9   11   13   15
[2,]    2    4    6    8   10   12   14   16
> dim(y)
[1] 2 8
> matrix(1:3, nrow = 2, ncol = 3, byrow = FALSE)
     [,1] [,2] [,3]
[1,]    1    3    2
[2,]    2    1    3
> matrix(1:3, nrow = 2, ncol = 3, byrow = TRUE)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    1    2    3
```

Узнать количество строк и столбцов матрицы можно, получив весь вектор размерностей с помощью функции `dim`, либо используя функции `nrow(x)` и `ncol(x)`. 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 для 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 функций `rownames(x)` и `colnames(x)`. `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, `A[5]` означает 5-й элемент матрицы `A`, если считать, что элементы перенумерованы по столбцам. Если `i` вектор, то `A[i]` означает выборку соответствующих элементов и т.д.

```
> m = matrix(1:6, nrow = 2, ncol = 3)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> m[1, 3]
[1] 5
> m[, 3]
[1] 5 6
> m[, c(1, 3)]
     [,1] [,2]
[1,]    1    5
[2,]    2    6
> m[-1,]
[1] 2 4 6
> A = matrix(c(23, 31, 58, 16), nrow = 2)
> rownames(A) <- c("petal", "sepal")
> colnames(A) <- c("length", "width")
> A
      length width
petal     23    58
sepal     31    16
> A["petal", "length"]
[1] 23
```

Для того, чтобы объединить несколько матриц в одну, используются функции `rbind(...)` (приписывает матрицы друг к другу снизу) и `cbind(...)` (приписывает матрицы друг к другу справа).

```
> A = matrix(1:4, nrow = 2, ncol = 2)
> B = matrix(5:8, nrow = 2, ncol = 2)
> cbind(A, B)
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
> rbind(A, B)
     [,1] [,2]
[1,]    1    3
[2,]    2    4
[3,]    5    7
[4,]    6    8
```

К матрицам можно применять арифметические операции, однако, они будут осуществляться покомпонентно. Для осуществления матричного умножения применяется оператор `%*%`.

Рассмотрим еще несколько функций, часто используемых при работе с матрицами.

- `t(A)`. Транспонирует матрицу.
- `diag(A)`. Возвращает вектор диагональных элементов матрицы, либо создает диагональную матрицы с указанными значениями диагонали.

```
> m = matrix(1:6, nrow = 2, ncol = 3)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> diag(m)
[1] 1 4
> diag(m) = c(11, 14)
> m
     [,1] [,2] [,3]
[1,]   11    3    5
[2,]    2   14    6
> a = diag(1:3, nrow = 3, ncol = 4)
> a
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    2    0    0
[3,]    0    0    3    0
```

- `det(A)`. Вычисляет определитель матрицы.
- `solve(A, b)`. Решает систему линейных уравнений $Ax = b$ с квадратной и невырожденной матрицей A. При этом `solve(A)` может применяться для отыскания обратной матрицы.
- `eigen(A)`. Вычисляет собственные чисела и векторы. Возвращает список, содержащий собственные числа и матрицу собственных векторов.
- `chol(A)`. Выполняет разложение Холецкого для симметричной положительно определенной матрицы.
- `qr(A)`. Выполняет QR-разложение матрицы. Возвращает список, содержащий полученные в результате разложения матрицы.
- `qr.solve(A, b, tol = 1e-7)`. Решает систему линейных уравнений через QR-разложение.
- `svd(A)`. Выполняет сингулярное разложение матрицы. Возвращает список, содержащий полученные в результате разложения матрицы.

3.3 Массив

Дальнейшим развитием идеи многомерности данных в языке R являются массивы, которые могут иметь произвольное количество размерностей. Работа с массивами почти идентична работе с матрицами. Содать массив можно либо изменив вектор размерностей вектора или матрицы с помощью функции `dim(x)`, либо используя функцию `array(data = NA, dim = length(data))`.

```
> 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

> x = 1:24
> dim(x) = c(2, 3, 4)
> x
, , 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
```

Для проверки является ли некоторая переменная вектором (не атомарным, а общим), матрицей или массивом используются фунции `is.vector(x)`, `is.matrix(x)`, `is.array(x)`.

3.4 Список

Как уже отмечалось выше, список является общим (generic) вектором, т.е. его элеметы могут иметь различный тип. Элементами списка могут быть в том числе векторы и другие списки. Для создания списка служит функция `list(...)`, принимающая на вход произвольное количество объектов, которые требуется объединить в список.

```
> L1 = list(1:5, c("a", "b"))
```

```
> L1
[[1]]
[1] 1 2 3 4 5

[[2]]
[1] "a" "b"

> L2 = list(rep(c(TRUE, FALSE), 3), L1)
> L2
[[1]]
[1]  TRUE FALSE  TRUE FALSE  TRUE FALSE

[[2]]
[[2]][[1]]
[1] 1 2 3 4 5

[[2]][[2]]
[1] "a" "b"
```

Для добавления новых элементов в список может служить функция конкатенирования списков `c(...)`. Применение к спискам способов индексирования, свойственных векторам, приведет к созданию нового списка, содержащего только выбранные элементы, даже если такой элемент всего один. Для доступа к элементам списка используется индексирование с помощью двойных квадратных скобок.

```
> x = list(1:5, c("a", "b", "c"), rep(c(FALSE, TRUE), 3),
    seq(0, 1, length = 11))
> x
[[1]]
[1] 1 2 3 4 5

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

[[3]]
[1] FALSE  TRUE FALSE  TRUE FALSE  TRUE

[[4]]
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

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

[[2]]
[1] FALSE  TRUE FALSE  TRUE FALSE  TRUE

> x[2]
[[1]]
[1] "a" "b" "c"

> x[[2]]
[1] "a" "b" "c"
```

Также, с помощью функции `names(x)` элементам списка можно присваивать имена, а затем использовать их в качестве индексов, указывая в квадратных скобках (одинарных или двойных) после имени списка, либо используя оператор `$`.

```
> x = list(1:5, c("a", "b", "c"), rep(c(FALSE, TRUE), 3),
    seq(0, 1, length = 11))
```

```
> names(x) = c("integers", "letters", "booleans", "doubles")
> x
$integers
[1] 1 2 3 4 5

$letters
[1] "a" "b" "c"

$booleans
[1] FALSE  TRUE FALSE  TRUE FALSE  TRUE

$doubles
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

> x[c("integers", "doubles")]
$integers
[1] 1 2 3 4 5

$doubles
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0

> x["letters"]
$letters
[1] "a" "b" "c"

> x[["letters"]]
[1] "a" "b" "c"
> x$letters
[1] "a" "b" "c"
```

3.5 Фактор

Факторы представляют собой структуру данных для хранения векторов категориальных данных (классов), т.е. величин, которые могут принимать значения из конечно, в общем случае, неупорядоченного множества. Факторы создаются с помощью функции `factor(x = character(), levels, labels = levels)`. 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
```

Для получения и изменения текстового вектора, содержащего имена уровней фактора служит функция `levels(x)`. Несмотря на то, что фактор не является вектором, изменяя с помощью фукнции `dim(x)` вектор размерностей, можно превратить фактор в фактор-матрицу и фактор-массив.

3.6 Фрейм

Фреймы данных (data frames) один из самых важных типов данных в R, позволяющий объединять данные разных типов вместе. Фрейм является специальной версией списка, где все элементы имеют одинаковую длину. Можно считать, что фрейм данных это 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(...)`, аргументами которой являются произвольное количество элементов (столбцов) фрейма. В качестве элементов фрейма данных могут выступать векторы, факторы, матрицы, списки или другие фреймы. При этом все векторы должны иметь одинаковую длину, а матрицы и фреймы одинаковое (такое же) число строк. Функция `data.frame(...)` просто собирает все данные вместе. Символьные векторы конвертируются в факторы. Остальные данные собираются во фрейм такими, какие они есть.

```
> a = matrix(1:8, nrow = 4, ncol = 2)
> b = c("a", "b", "c", "a")
> d = (1:4 %% 2 == 0)
> e = factor(c("soft", "hard", "soft", "medium"))
> f = data.frame(a, b, d, e)
> f
  X1 X2 b     d      e
1  1  5 a FALSE   soft
2  2  6 b  TRUE   hard
3  3  7 c FALSE   soft
4  4  8 a  TRUE medium
> f[[1]]
[1] 1 2 3 4
> f[["b"]]
[1] a b c a
Levels: a b c
> f$d
[1] FALSE  TRUE FALSE  TRUE
```

С помощью функции `colnames(x)` можно изменить имена столбцов фрейма, с помощью `rownames(x)` – имена строк.

Фрейм данных идеально подходит для хранения выборки данных, где каждому столбцу соответствует некоторая переменная или признак, а каждой строке отдельный прецедент.

Для загрузки наборов данных из файла в фрейм может быть использована функция `read.table(file, header = FALSE, sep = "", ...)`, которой на вход подается путь к текстовому файлу с данными, в котором значения в каждой строке разделены символом `sep`. Параметр `header` позволяет указать, следует ли интерпретировать первую строку файла, как имена столбцов таблицы.

4 Управляющие конструкции

Приведем краткое описание условных конструкций и циклов языка R.

4.1 if/else

```
if (condition)
{
    # expression
}

if (condition)
{
    # expression
}
```

```
else
{
    # alternative expression
}
```

Условие `condition` в условной конструкции `if`, как и в циклах, может давать в качестве результата логический вектор, однако, в рассмотрение принимается только первая его компонента. Также могут быть полезными функции `any(x)` и `all(x)`, которые, принимая на вход логический вектор, возвращают значение `TRUE` тогда и только тогда, когда хотя бы один или все элементы вектора равены `TRUE` соответственно.

4.2 for

```
for (i in I)
{
    # expression
}
```

Переменная `i` последовательно принимает все значения из вектора `I`.

Преждевременный выход из любого цикла (не обязательно `for`) осуществляется командой `break`. Для прерывания текущей итерации и перехода к следующей служит команда `next`.

4.3 while

```
while (condition)
{
    # expression
}
```

4.4 repeat

```
repeat
{
    # expression
}
```

Данная конструкция реализует бесконечный цикл, выход из которого можно осуществить с помощью команды `break`.

5 Функции

Функции в языке R определяются с помощью ключевого слова `function`, вслед за которым в круглых скобках идет перечисление формальных агрументов функции. Аргументам можно назначать значения по умолчанию, как выражения, зависящие, в том числе, и от других формальных аргументов. При этом вычисление данного выражения будет производиться только, если значение этого агрумента потребуется при выполнении тела функции. Тело функции идет после указания списка ее формальных аргументов и должно быть заключено в фигурные скобки если содержит более одного выражения. Значением, возвращаемым функцией является либо результат последнего выражения в теле функции, либо аргумент функции `return(x)`, прерывающей выполнение функции.

Как правило, определенная описанным выше способом функция присваивается некоторому имени с помощью опреатора присваивания. Анонимные функции могут быть использованы, как фактические аргументы других функций.

```
> sq = function(x)
+ {
+     x * x
+ }
>
> fun = function(x, a = x * x)
+ {
+     return(x + a)
+ }
```

Все аргументы передаются в функцию по значению, а, следовательно, не могут быть изменены в ней. Для того, чтобы вернуть несколько однотипных значений, следует возвращать вектор, разнотипных – список.

Вызов функции осуществляется путем указания ее имени с последующим перечислением фактических аргументов в круглых скобках, как это делалось для всех рассмотренных функций. Следует отметить, что помимо указания всех аргументов в порядке, указанном при определении функции, часть или все аргументы могут быть указаны в именованном виде `argumentName = argumentValue`, при этом порядок следования не важен.

6 Формулы

Зачастую в ходе статистического моделирования необходимо выразить зависимость одной величины от другой и описать характер данной зависимости. В языке R для это используются специальные объекты, называемые формулами. Для определения формулы используется оператор `~`, синтаксис которого выглядит следующим образом: `y ~ model`. Слева от оператора `~` указывается имя зависимой переменной, а справа выражение специального вида – модель зависимости. Например, формула `y ~ x1 + x2` определяет линейную зависимость `y` от `x1` и `x2`: $y = a_0 + a_1x_1 + a_2x_2$. Все переменные, используемые в модели, должны быть либо числовыми векторами, либо факторами. При этом логические векторы преобразуются в факторы.

Рассмотрим правила формирования модели формулы. Знаки `+` и `-` определют вхождение или исключение определенной переменной как линейного члена в модель, т.е. в данном случае семантика данных операций отлична от просто арифметического сложения и вычитания значений переменных. При необходимости арифметические операции могут быть использованы в модели с помощью функции `I(x)`, которая позволяет интерпретировать свой аргумент, как обычное выражение языка R. Также преобразования (не только арифметические) переменных могут быть предварительно заданы отдельными функциями, которые можно в последствие использовать в модели формулы, например, `log(x)` или `f(x1, x2)`.

Оператор `:` при использовании в модели формулы обозначает взаимное влияние нескольких переменных. Например, если `x1` и `x2` числовые векторы, то член `x1:x2` равносилен `I(x1 *x2)`. В случае, если `x1` или/и `x2` является фактором, то отдельно рассматриваются все его уровни. Например, если `x1` – фактор с двумя уровнями `a` и `b`, а `x2` – числовой вектор, то `x1` заменяется на два числовых 0-1 вектора `z1 = as.numeric(x1 == "a")` и `z2 = as.numeric(x1 == "b")` и модель эквивалентна `z1:x2 + z2:x2`.

Оператор `*` обозначает взаимодействие и главные эффекты, т.е. влияние самих переменных и их совместное влияние. `x1 * x2` эквивалентно `x1`

`+ x2 + x1:x2`, а `x1 * x2 * x3` эквивалентно `x1 + x2 + x3 + x1:x2 + x1:x3 + x2:x3 + x1:x2:x3`.

Оператор `^` обозначает главные эффекты и все взимодействия вплоть до указанного порядка. `(x1 + x2 + x3)^ 2` эквивалентно `x1 + x2 + x3 + x1:x2 + x1:x3 + x2:x3`.

7 Пакеты расширений

Одной из причин популярности языка R является обилие бесплатных пакетов (packages), расширяющих базовые возможности языка. Основным репозиторием пакетов расширений является репозиторий CRAN (http://cran.r-project.org), на настоящий момент насчитывающий более 4000 пакетов.

Для использования определенного пакета, требуется установить его со всеми зависимостями и подключить. Для того, чтобы установить пакет, можно скачать его из репозитория в бинарном виде или в виде исходного кода (в этом случае придется выполнять сборку пакета вручную), а затем установить с помощью функции `install.packages(pkgs, lib)`, первым аргументом которой является путь к zip-файлу с пакетом в бинарном виде, а вторым – путь к директории (библиотеке), в которую будет установлен данный пакет. Список известных библиотек можно посмотреть, вызвав функцию `.libPaths()`. Если при вызове `install.packages` параметр `lib` не был указан, то используется первая компонента вектора `.libPaths()`. Другим способом установки пакета является его автоматическая закачка из репозитория, для этого в качестве параметра `pkgs` функции `install.packages` требуется указать только имя пакета, который в случае удачного поиска в репозитории будет скачан и установлен вместе со всеми зависимостями.

Посмотреть список установленных пакетов можно с помощью функции `library()`. Чтобы загрузить пакет для использования, необходимо вызвать функцию `library(package)` или `require(package)`, где `package` – имя пакета. Чтобы посмотреть краткую справку об установленном пакете можно вызвать функцию `library(help = package)`.

Обновить установленные пакеты можно функцией `update.packages()`.

8 Графика

Для изображения графиков в R есть функция `plot(x, y, ...)`. Здесь `x` – вектор значений абсцисс и `y` – вектор значений ординат.

Функция `plot` допускает множество дополнительных параметров. Приведем описание некоторых из них:

- `log = "x"` или `log = "y"` или `log = "xy"` делает соответственно масштаб по оси абсцисс, ординат или по обеим осям логарифмическим;
- `type` 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` и `ylab` позволяет указать подписи к оси абсцис и ординат соответственно;

- `main` и `sub` создают надписи сверху и снизу графика;
- `col` задает цвет графика. Некоторые возможные значения: `"blue"`, `"red"`, `"green"`, `"cyan"`, `"magenta"`, `"yellow"`, `"black"`. Можно задать цвет rgb-вектором функцией `rgb(r, g, b)`, где `r`, `g` и `b` от 0 до 1.
- `lty` определяет стиль линии. Некоторые возможные значения `"blank"` (нет линии), `"solid"` (сплошная линия), `"dashed"` (пунктирная линия), `"dotted"` (точечная линия), `"dotdash"` (точка-тире).

Следующие функции 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.
- `text(x, y = NULL, labels = seq_along(x), ...)` добавляет текст. По умолчанию он центрируется вокруг точки (x, y). Указанные параметры могут быть векторами. В этом случае будет размещено несколько надписей.
- `abline(k, b, ...)` рисует прямую $y = kx + b$.
- `abline(h = y, ...)` и `abline(v = x, ...)` отрисовывают горизонтальные и вертикальные прямые соответственно.
- `legend(x, y = NULL, legend, ...)` добавляет легенду в указанную точку.

Рассмотрим пример.

```
> x = seq(0, 2 * pi, length = 250)
> plot(x, sin(x), col = "red", type = "l", main = "sine and
    cosine", lty = "dashed")
> lines(x, cos(x), col = "blue")
> legend(0, -0.75, legend = c("sine", "cosine"), lty =
    c("dashed", "solid"), col = c("red", "blue"))
```

Результат выполнения данного набора команд приведен на рис. 1.

9 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 0.3. Подсчитайте статистические оценки мат. ожидания, стандарного отклонения, квантилей уровней 0.95 и 0.99 с и без использования встроенных функций. Сравните результат. Исследуйте отклонение статистического мат. ожидания от 1 при росте N ($N = 1000, 2000, 4000, 8000, 16000, 32000$).
2. Создайте фрейм данных из $N = 20$ записей со следующими полями: $Nrow$ – номер записи, $Name$ – имя сотрудника, $BirthYear$ – год рождения, $EmployYear$ – год приема на работу, $Salary$ – зарплата. Заполните данный фрейм данными так, что $Nrow$ изменяется от 1 до N,

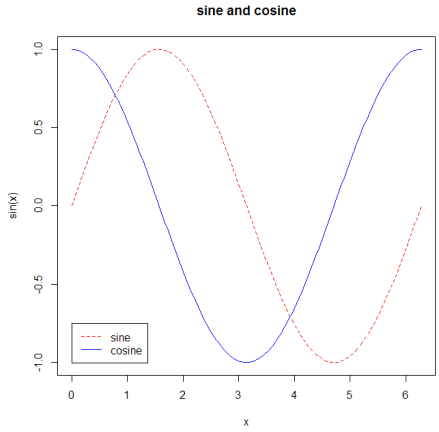

Рис. 1:

Name задается произвольно, $BithYear$ распределен равномерно (случайно) 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 год 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 x, разделяет полученный отрезок $[x_{min}; x_{max}]$ на k равных интервалов и подсчитывает число элементов вектора, принадлежащих каждому интервалу.

   Постройте график, где по оси абсцисс откладываются 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, выборка реализаций экспоненциально распределенной случайной величины с параметром $\lambda = 500$. Приближение какого графика мы получаем в итоге при большом числе точек и числе разбиений?

4. Пусть дан набор точек в $x_i \in \mathbb{R}^d$, $i = 1, 2, \ldots, n.$, каждой из которых поставлено в соотвествие некоторое вещественное число y. Реализуйте метод наименьших квадратов, находящий гиперплоскость, обеспечивающую наименьшую невязку по y. Для случая $d = 1$ нарисуйте график, содержащий исходные точки и найденную прямую.