Федеральное государственное образовательное бюджетное учреждение
высшего профессионального образования
«ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»
(Финансовый университет)

Департамент анализа данных, принятия решений
и финансовых технологий

Дисциплина «Программирование в среде R»

П.Б. Лукьянов

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ
ЛАБОРАТОРНОЙ РАБОТЫ № 2

Основы и простейшие команды языка R

Для студентов, обучающихся по направлению подготовки
«Прикладная информатика»
(программа подготовки бакалавра)

Москва 2020

Цель лабораторной работы заключается в изучении основ и особенностей языка R, знакомстве с простейшими командами и вычислениями на R.

Язык R относится к высокоуровневым языкам программирования, что подразумевает удобство программирования, работу с различными структурами данных, а также наличие типовых конструкций языка для построения гибких алгоритмов при решении разнообразных задач. В этом язык R похож на остальные современные высокоуровневые языки: C, Java, Python, Visual Basic и т.д.

Вместе с тем, у языка есть ряд особенностей, которые характерны только для R, о которых будет сказано дополнительно.

1. Комментарии в программе

Код, который вам понятен сегодня, но в котором нет комментариев, через неделю превращается в сложный кроссворд. Поэтому общепринятой практикой среди программистов является документирование всех ключевых действий программы посредством написания комментариев. Комментарии пишутся программистом для удобства сопровождения программы и ее возможного последующего развития. Всегда оставляйте комментарии, какой бы простой разрабатываемая программа вам не казалась.

Комментарии в языке R начинаются со знака # и продолжаются до конца строки. Текст комментариев может быть на любом языке, может занимать любое количество строк.

Не нужно беспокоиться о том, что комментарии увеличивают программу или замедляют работу. При запуске программы ее код проходит предварительную обработку, и из него автоматически удаляются все пробелы и комментарии, и лишь затем программа выполняется.

2. Консольный ввод команд

В среде R в строке ввода после приглашающего значка > как правило, вводится одна команда. После нажатия Enter эта команда сразу выполняется и выводится результат выполнения этой команды. Например:

```
> 1 + 2  + 4   # наша команда  – выполнить сложение
[1] 7          # ответ среды R
```

В строке можно ввести несколько команд, если разделять их точкой с запятой. В случаях, когда команда длинная (например, вызывается функция с большим количеством параметров), можно нажать Enter и продолжить ввод на следующей строке. В этом случае слева вместо значка > появляется +.

Клавиши «Стрелка вверх», «Стрелка вниз» позволяют перемещаться по ранее введенным командам, а клавиши «Стрелка влево», «Стрелка вправо» используют для редактирования текущей команды. История всех команд сохраняется во время работы, поэтому для повторного вызова набранной ранее команды достаточно нажать клавишу «Стрелка вверх» нужное число раз.

Проверим работу R в режиме калькулятора. Наберите в строке ввода несколько простейших математических выражений, в конце нажмите Enter:

```
5 * 12 / 3 + 0.1
sin(54); cos(45); sin(54) + cos(45);
sqrt(abs( sqrt(-9 * (-5)) / (-27.7^2)))
11 %/% 3; 11 %% 3
```

Обратите внимание, что в качестве десятичного разделителя используется точка; количество пробелов в выражениях может быть любым, имена функций обычно пишутся с маленькой буквы. Если в командной строке есть разделитель точка с запятой, то результаты выводятся независимо друг от друга.

В выражении a %% b оператор %% означает остаток от деления a на b. Выясните, что означает операция a %/% b.

Язык R чувствителен к регистру, поэтому будьте внимательны при вводе команд и функций. Так, функции makeSpecialActions() и MakeSpecialActions() различны. Аналогично, переменные sumXY и sumXy – это разные переменные.

В режиме калькулятора результат вычислений выводится на экран, но нигде не сохраняется. Для того, чтобы результаты вычислений сохранить в памяти компьютера и затем обращаться к этим результатам, используют переменные и константы.

3. Имена переменных и констант в языке R

Проще всего переменные представлять как ящики разного размера, в которых могут храниться различные данные; все ящики подписаны, у каждого ящика есть имя. Другими словами, переменная – это именованная ячейка в памяти компьютера, выделенная для хранения некоторой величины.

В любом языке программирования используются переменные и константы. В R имена переменных и констант могут быть как на латинице, так и на кириллице, ошибки здесь нет. Но если предположить, что в будущем ваши программы будут включены в международные проекты, то для всех имен рекомендуется использовать символы английского языка.

Переменные используются для хранения данных, значения которых могут быть изменены, в этом их отличие от констант. Переменные и константы однозначно определяются своими именами, и при задании имен в R есть свои правила.

Имя – это непрерывная последовательность букв, цифр, точек и символов подчеркивания. Начинать имя переменной с цифры или символа подчеркивания нельзя. В имени переменной могут использоваться точка и

символ подчеркивания, но в настоящее время в программировании отказываются от использования подчеркивания в пользу точки в качестве смыслового разделителя.

Например, точку удобно использовать в таком сложном имени переменной: local.param.for.regression. Без точки сложное имя принято конструировать так: localParamForRegression. Заглавные буквы для первого символа переменной обычно не используют.

Если имя начинается с точки, то второй символ не может быть цифрой. Как уже отмечалось, прописные и строчные символы различаются: res, Res, RES – это разные имена.

Корректные имена в R:

result, Sum, .everage.value, is_true, File29_5

Недопустимые имена в R:

1ff, min@l, 14days, _no_, TRUE, .02

Еще одно ограничение при конструировании имен переменных и констант заключается в следующем: для имени нельзя использовать **зарезервированные** слова, т.е. слова, которые уже имеют вполне определенный смысл в языке R. Ниже представлен список зарезервированных слов:

if, else, repeat, while, function, for, in, next, break, TRUE, FALSE, NULL, Inf, NaN, NA, NA_integer_, NA_real_, NA_complex_, NA_character_

Часть этих слов (if, else, repeat, while, function, for, in, next, break) используется в управляющих конструкциях языка, таких как проверка условий, циклы, функции, заданные Пользователем.

TRUE и FALSE – логические константы

NULL представляет собой «ничто». NULL характеризует пустое, неопределенное значение. Например, при обращении к несуществующему элементу списка мы получим в качестве результата NULL.

Inf – бесконечность (infinity); как правило, результат деления числа на ноль, бесконечность может быть как положительная (1/0), так и отрицательная (-1/0).

NaN – неопределенный результат (Not a Number). NaN возвращается при операциях над числами, результат которых не определён (не является числом).

NA – отсутствующее значение (Not Available). NA возникает, если значение некоторого объекта не доступно или не задано. NA используется для замещения отсутствующих значений, например, при анализе данных. Так, при обращении к несуществующему элементу массива мы получим в качестве результата NA.

4. Динамическая и статическая типизация

Язык R является динамически типизированным языком (как и Python, Objective-C, Ruby, PHP, Perl, JavaScript и др.), в отличие от языков со статической типизацией (C++, C#, Java, Паскаль и др).

Идея динамической типизации заключается в том, что пока переменной не присвоено какое-либо значение, тип этой переменной не определен. Только получив значение, например строковое, переменная приобретает и тип. В нашем случае тип переменной будет определен как символьный. В другом месте программы этой же переменной может быть присвоено числовое значение, соответственно тип переменной изменится со строкового на числовой.

Динамическая типизация дает программисту дополнительную свободу в написании программ, так как одну и ту же переменную можно многократно использовать в разных местах программы в разных

контекстах. В идее динамической типизации есть и минус: о возможных ошибках в выражениях с переменными мы узнаем только при выполнении программы.

В отличие от динамической типизации статическая типизация требует определения типа переменной сразу при ее объявлении, и в дальнейшем тип переменной не может быть изменен. Такой подход повышает надежность программ и позволяет выявить ряд ошибок сразу, до запуска программы, на этапе автоматических проверок перед преобразованием рукописных комманд в машинные коды.

Одним из недостатков статической типизации считается сложность работы с базами данных и внешними структурами данных, так как в одних и тех же полях могут храниться данные разных типов, и при их считывании в переменные определенного, фиксированного типа будут возникать ошибки.

5. Типы переменных и констант

Тип переменной или константы определяется типом значения, присвоенного этой переменной или константе. В R выделяют три числовых типа:

- целые (**integer**)
- действительные (**double**)
- комплексные (**complex**)

Все числовые типы относятся к типу **numeric**, т.е. переменные типа **integer**, **double** и **complex** одновременно являются переменными типа **numeric**.

Кроме числовых типов, переменная или константа может быть логической (**logical**) или символьной (**character**). Логическая переменная может принимать только два значения TRUE (истинно) или FALSE (ложно). Используют и сокращенные обозначения: T или F.

Как уже отмечалось, TRUE и FALSE – зарезервированные слова в языке R, поэтому использование их в качестве имен переменных приведет к ошибке, но использование T и F в качестве имен переменных к ошибке не приводит.

Символьный тип предназначен для хранения одиночных символов и их последовательностей (строк).

Числовые константы

Числовые константы, за которыми следует L, считаются целыми, а те, за которыми следует i, считаются комплексными. Если после числа никакого символа нет, число считается типа **double**, это значение по умолчанию.

Если перед числом стоит 0x или 0X, число считается шестнадцатеричным.

Символьные константы

Символьные константы (**character**) задаются при помощи одинарных кавычек (') или двойных кавычек (") .

Примеры:

'Просто строчка', "17L * 0.1”, " ", 'abc cba'

Логические константы

Логические константы (**logical**) задаются присвоением им значений TRUE или FALSE.

Встроенные константы

Ниже приведены некоторые из встроенных констант языка R.

Число пи. Наберите pi и нажмете Enter, должно получиться следующее:

> pi

3,141593

Названия месяцев на английском языке хранятся в специальном массиве:

>month.name

"January" "February" "March" "April" "May" "June"

"July" "August" "September" "October" "November" "December"

Сокращения названий месяцев:

> month.abb

"Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"

Переопределение констант

Константы легко переопределить. Так, присвоим pi другое значение, например, 99 и заодно познакомимся с оператором присваивания в R. Чтобы объявить переменную или константу и задать ей какое-либо значение, используют оператор присваивания; это конструкция, состоящая из знаков «меньше» и «тире»: <-.

Наберите следующую строчку, в конце нажмите Enter.

> pi <- 99

> pi

[1] 99

Таким образом, с точки зрения языка, константы ничем не отличаются от переменных, всю логику создания и использования констант и переменных определяет сам программист.

Для того, чтобы отличать константы от переменных, программисты часто используют дополнительные соглашения об именовании констант: в именах констант принято использовать только заглавные буквы и символ подчеркивания, например

MIN_LEVEL

MAX_ATTEMPT_AMOUNT

TAX_VALUE

Соответственно, значения констант определяются в программе один раз, как правило, в самом начале, в специальном разделе инициализации констант и переменных программы.

Определение и явное преобразование типов переменных и констант

Узнать, к какому типу принадлежит переменная или константа, можно с помощью нескольких функций языка:

typeof(имя), где имя – имя переменной или константы

is.numeric(имя)

is.integer(имя)

is.double(имя)

is.numeric(имя)

is.logical(имя)

is.character(имя)

is.finite(имя)

is.infinite(имя)

is.na(имя)

is.nan(имя)

Четыре последние функции предназначены для определения того, является ли значение переменной соответственно конечным значением, бесконечностью, отсутствующим значением, неопределенным результатом. Эти функции, как правило, используются при обработке результатов экспериментов и при анализе данных.

Можно явно изменить тип переменной, в результате чего значение, которое она содержит, будет преобразовано к новому типу. Функции для преобразования типов следующие:

as.numeric(имя)

as.integer(имя)

as.double(имя)

as.numeric(имя)

as.logical(имя)

as.character(имя)

6.Операторы присваивания

Рассмотрим пример:

q <- 5

Объявляется переменная q, ей присваивается значение 5. q стала переменной действительного типа (почему?).

b <- 5.1

Создается переменная b действительного типа, ее значение 5.1

Вместо присваивания с помощью <- можно пользоваться как традиционным равенством =, так и присваиванием в другую сторону с помощью ->:

x = 87

87 -> x

Проверьте, что задавать значения переменным можно всеми этими способами. Общепринятым оператором присваивания в R является <-, именно им и будем пользоваться в дальнейшем.

Одно значение можно присвоить сразу нескольким переменным:

a <- b <- c <- 5.05

В языке есть еще два редко используемых оператора присваивания:

x <<- 10

10 ->> x

Присваивание значений переменным с использованием этих операторов используется в функциях, создаваемых Пользователем. Такое присвоение позволяет переопределять значения глобальных переменных (переменных, видимых во всех частях программы), находясь внутри функции. Результат использования этих операторов не такой очевидный,

что связано с понятием глобальной среды, поэтому оставим этот вопрос для рассмотрения в разделе, посвященном функциям, создаваемым Пользователями.

Мы можем создать переменную, мы можем ее и удалить. Удаление выполняется с помощью функции rm(имя_переменной). Создайте несколько переменных разных типов, проверьте значения и тип переменных, затем удалите некоторые из них.

Как убедиться в том, что переменная действительно была удалена?

7. Арифметические операторы

Для операций с числами и числовыми переменными используются обычные операторы сложения, вычитания, умножения, деления и возведения в степень. Кроме этого, есть операция целочисленного деления (%/%) и операция получения остатка от деления (%%). Операции имеют обычный приоритет, т.е. сначала выполняется возведение в степень, затем умножение или деление, затем сложение или вычитание. Если используются круглые скобки, то операции в скобках выполняются в первую очередь. Все арифметические операторы представлены в Таблице 1.

Таблица 1. Арифметические операторы

+	Сложение
–	Вычитание
*	Умножение
/	Деление
^	Возведение в степень
%/%	Целочисленное деление
%%	Остаток от деления

Самостоятельно проверьте работу этих операторов и объясните полученный результат.

8. Условные операторы и неявное преобразование типов

Условные операторы предназначены для проверки истинности различных условий и представлены в Таблице 2. Создайте несколько числовых, логических и символьных переменных и проверьте работу этих операторов. Обратите внимание, что сравнивать между собой можно не только числовые переменные, но и переменные других типов. Результат сравнения всегда принимает значение TRUE или FALSE.

Таблица 2. Условные операторы

<	Меньше
>	Больше
<=	Меньше или равно
>=	Больше или равно
==	Равно
!=	Не равно

При написании программ возможны ситуации, когда в выражениях сравнения участвуют переменные разных типов. В этом случае при выполнении программы автоматически происходит неявное преобразование типов. Делается это для того, чтобы операция сравнения была корректной и выполнялась с переменными одного типа.

Рассмотрим пример. Наберите на компьютере следующие команды, после каждой команды нажимайте Enter:

```
a <- 29
b <- '29'
c <- a == b
```

Чему равно значение переменной c? Наберите c и нажмите Enter, убедитесь, что значение c равно TRUE. Из этого результата следует, что

1) или числовое значение переменной a было преобразовано к строке ‘29’
2) или строковая переменная b была преобразована в число 29.

Выясним, какой вариант верный. Рассмотрим второй пример. Наберите:

```
a <- 28
b <- '30 - 2'
c <- a<b
```

Чему равно значение c? Какой вывод отсюда можно сделать? Проверьте свои заключения, выполнив третий пример:

```
a <- 28
b <- '20 + 10'
c <- a<b
```

Чему равно c?

9. Логические операторы

С помощью логических операторов можно тестировать на истинность не одно выражение, а сразу несколько. Для связывания нескольких условий в одно логическое выражение используются логические операторы (Таблица 3).

Таблица 3. Логические операторы

!	Логическое НЕ (NOT)
&	Поэлементное логическое И (AND)
&&	Сокращенное логическое И (AND)

|	Поэлементное логическое ИЛИ (OR)
||	Сокращенное логическое ИЛИ (OR)

Например, требуется, чтобы определенная часть программы была выполнена, только если совпадут все требуемые условия – в этом случае необходимо использовать логическое И; или при анализе значений нескольких переменных требуется, чтобы хотя бы одно из значений находилось в нужном диапазоне – в логическом выражении нужно использовать логическое ИЛИ.

Как видно из таблицы, для логического И и логического ИЛИ существует по два оператора. Важно понимать разницу в их работе. Отличие заключается в способе обработки операндов (сравниваемых переменных). Если сравниваются скалярные значения, то разницы в работе операторов нет, сравните два примера

Пример 1. Используем поэлементное ИЛИ:

```
q1 <-2
q2 <-0
q3 <- (q1 == 2) | (q2 ==2)
```

Чему равно q3?

Пример 2. Используем сокращенное ИЛИ:

```
q1 <-2
q2 <-0
q3 <- (q1 == 2) || (q2 ==2)
```

Чему равно q3?

Отличие в операторах проявится, если вместо скалярных переменных сравниваются вектора. Что такое вектор и как в языке R работать с векторами – это тема следующих разделов. Сейчас же нужно понять, что сравнивать можно не только скалярные значения. Рассмотрим

примеры 3 и 4, где переменны q1 и q2 – вектора (последовательности чисел).

Пример 3. Используем поэлементное ИЛИ:

```
q1 <-(2:4)        # вектор из элементов 2, 3, 4
q2 <-(0:2)        # вектор из элементов 0, 1, 2
q3 <- (q1 == 2) | (q2 ==2)
```

Чему равно q3?

Пример 4. Используем сокращенное ИЛИ:

```
q1 <-(2:4)
q2 <-(0:2)
q3 <- (q1 == 2) || (q2 ==2)
```

Чему равно q3?

Как видно из результатов выполнения примеров 3 и 4, значения q3 различны. Именно в этом смысл использования сокращенного и поэлементного И.

Контрольные вопросы и задания

1. Перечислить по памяти зарезервированные слова языка R. Сколько их всего? Объяснить смысл NULL, Inf, NA, NaN. Получить эти значения несколькими способами в результате выполнения каких-либо операций.
2. Создать десять переменных разных типов, проверить их тип с помощью функции typeof(). Проверить с помощью этой функции типы следующих констант: 29, 23i, -34L, 2/3, 4/2, 0xA, 0XbL – 120L, 0XbL – 120, 0XbL * 17. Объяснить полученные результаты.
3. Проверить работу арифметических операторов и приоритет выполнения операций в сложных выражениях. Выполнить серию из 10 вычислений по различным самостоятельно придуманным формулам. Использовать все операторы из таблицы 1.

Выяснить правило, по которому выполняется расчет, если в формуле используется целочисленное деление и остаток от деления. Убедиться, что значения выражений дают предсказуемый вами результат.

4. Исследовать правила неявного преобразования типов, выполнив примеры, аналогичные примерам из п.8. Определить правила преобразования для переменных следующих типов:

   integer и double

   integer и logical

   logical и character

   double и logical

   double и character

   По результатам исследования сформулировать общее правило преобразования типов в R.

5. Исследовать различие в работе логических операторов «Поэлементное И» и «Сокращенное И», для чего подготовить примеры, аналогичные примерам из п.9.

6. Последовательно выполнить операции:

   3/7

   3/7 – 0.4285714

   Объяснить полученный результат.

7. Последовательно выполнить операции:

   sqrt(2)*sqrt(2)

   (sqrt(2)*sqrt(2))-2

   Объяснить полученный результат.