Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное
образовательное учреждение высшего образования

ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

А. В. Коросов
В. В. Горбач

ПРАКТИЧЕСКОЕ ВВЕДЕНИЕ В СРЕДУ R

Учебное электронное пособие для обучающихся по направлениям подготовки «Биология» и «Экология и природопользование»

Петрозаводск
Издательство ПетрГУ
2021

УДК 574:004
ББК 28.081
К686

Издается по решению редакционно-издательского совета
Петрозаводского государственного университета

Рецензенты:
Г. С. Розенберг, член-корреспондент РАН, доктор биологических наук;
В. Н. Якимов, доктор биологических наук

Коросов, Андрей Викторович.

К686 Практическое введение в среду R [Электронный ресурс] : учебное электронное пособие для обучающихся по направлениям подготовки «Биология» и «Экология и природопользование» / А. В. Коросов, В. В. Горбач ; М-во науки и высш. образования Рос. Федерации, Федер. гос. бюджет. образоват. учреждение высш. образования Петрозавод. гос. ун-т. – Электрон. дан. – Петрозаводск : Издательство ПетрГУ, 2020. – 1 электрон. опт. диск (CD-R) ; 12 см. – Систем. требования : PC, MAC с процессором Intel 1,3 ГГц и выше ; Windows, MAC OSX ; 256 Мб ; видеосистема : разрешение экрана 800 × 600 и выше ; графический ускоритель (опционально) ; мышь или другое аналогичное устройство. – Загл. с этикетки диска.

ISBN 978-5-8021-3505-1

Учебное пособие призвано способствовать овладению навыками профессиональной работы в среде R в процессе создания программ для обработки биологических и экологических данных на компьютере. В доступной форме изложены основы программирования на языке R, принципы организации баз данных и работы с ними. Кратко рассмотрены способы построения диаграмм и основные методы статистической обработки, начиная с описательной статистики и заканчивая компонентным анализом и имитационным моделированием. Особое внимание уделено способам формирования выборок и подготовки их к статистической обработке в среде R.

УДК 574:004
ББК 28.081

ISBN 978-5-8021-3505-1

Предисловие

Авторы этой книги – зоологи, имеющие продолжительный опыт применения математических методов в научном поиске. Они апробировали множество разных программ и методов, написали 12 учебников и методических пособий по биометрии. Кроме того, они ведут несколько биометрических курсов у студентов эколого-биологического направления, в частности, математические методы в биологии и экологии, специальные методы биометрии, математическое моделирование биологических процессов и др. Их многолетний опыт показывает, что среда программы R служит эффективным средством обработки биологической информации. Авторы стремились создать руководство, способное сократить время для освоения этой программы студентами. Одни из лучших произведений, обучающих R, написаны В. К. Шитиковым (см. список рекомендуемой литературы). В них можно найти ответы практически на все вопросы, интересующие биологов относительно методов обработки биологической информации. В то же время эти книги достаточно сложны для понимания начинающими; на наш взгляд, необходимы адаптивные пособия, которые постепенно подводили бы читателей к профессиональному использованию программы R. К сожалению, многие учебные издания по этому языку включают огромное количество детальной избыточной информации. Они незаменимы, но не позволяют овладеть навыками программирования для решения конкретных биометрических задач. С другой стороны, книги по программированию на R основаны на математических примерах, что очень усложняет процесс овладения языком биологами. Итак, ключевые слова нашей работы: программирование на R, биологические задачи, простота. Наше пособие должно рассматривать пути решения типичных биологических (экологических) задач, нести черты не справочного пособия, но рекомендации по их решению, быть достаточно компактным и простым для студентов старших курсов, специалистов-биологов и экологов.

ВВЕДЕНИЕ

 нужен тем, кто хочет обрабатывать данные максимально быстро и полно. В пакетах программ R (packages) реализованы практически все мыслимые процедуры количественной обработки, в том числе биологических и экологических данных (R Core Team, 2012). А то, чего нет, можно сделать самому. Авторы пока еще не решили для себя, что быстрее – найти пакет с нужной функцией или написать свой код?

Среда R состоит из двух частей – это:
1) набор готовых программ (функций) для разнообразной математической обработки;
2) язык программирования.

Среди разнообразных математических функций биологу, прежде всего, интересен статистический анализ данных. Работа с функциями, которые рассчитывают среднюю, дисперсию, линейную регрессию, дисперсионный анализ, компонентный анализ, кластеризацию и многое другое, уже описана нами на биологических примерах в нескольких учебных пособиях (Ивантер, Коросов, 2015; Коросов, Горбач, 2015; Коросов, Горбач, 2017). В этой книге будут разобраны некоторые полезные биологам *методы программирования на R* и необходимые для этого средства языка R. В отличие от специальных статистических пакетов (Statgraphics, Statistica и др.) команды R набираются с клавиатуры в виде командной строки. Значит, для выполнения любого вида обработки данных необходимо понять структуру языка, выучить основные правила синтаксиса и овладеть некоторыми навыками для составления своих программ.

Типичная программа для обработки биологических данных состоит из трех частей:
1) извлечение из обширной базы данных, записанной в файле, относительно небольшой выборки для последующей статистической обработки (чтение, сортировка, правка, унификация);
2) выполнение статистической обработки этой выборки с помощью стандартных функций R (расчет средних, регрессионный анализ и пр.);
3) вывод результатов – построение диаграмм и таблиц со статистическими оценками.

Готовые статистические функции и программирование связаны между собой, во-первых, прямой связью: пользователь-биолог пишет программу и включает в нее функции статистической обработки. Существует и более сложная обратная связь. Базовые функции, включённые в пакеты Base, Stats, Tools, Utils и др. (организация массивов, операции присваивания, арифметические действия, основные статистические процедуры и пр.), *написаны на языке C++*; они устанавливаются на компьютер в мо-

мент инсталляции программы R. Кроме этого набора *пользователи по всему миру написали на языке R* множество дополнительных пакетов (например, для дискриминантного анализа – MASS). Такие пакеты количественной обработки нужно с помощью команд Главного меню специально загрузить из интернета (Пакеты \ Установить пакет(ы)) и включить (Пакеты \ Включить пакет). Любой пользователь может запрограммировать свою функцию и пополнить общую глобальная копилку функций (если ее одобрят в центе унификации в Австрии). На сегодняшний день в сети представлено более 10 000 пакетов, однако мы рассматриваем только базовые, которые устанавливаются при инсталляции R.

Программа

Программа – это реализация алгоритма (определенной последовательности действий), формальное описание некоторого процесса в среде компьютерного языка. Язык программирования R позволяет алгоритмический процесс представить в виде структурной записи. Вместо выполнения серии идущих друг за друга команд мы можем в одной строке сконструировать выражения, реализующие эти команды.

Элементарная операция записывается как выполнение некой **функции** над неким объектом (с именем *массив*) при заданных условиях (аргумент 1, 2...): **функция** (*массив*, аргумент 1 = значение, аргумент 2 = значение, ...). В результате объект меняет свои свойства, становится другим объектом. Его можно использовать в дальнейших расчётах (преобразованиях), в том числе вывести на печать в окне консоли, построить диаграмму в окне графики, записать в файл. Вместо текста «результатом выполнения функции является...» часто используют выражение «функция возвращает...» новый объект.

Программирование на R состоит в построении алгоритмов в форме «матрешек» из вложенных функций, массивов, аргументов и констант. Именованный исходный массив данных помещается в качестве аргумента в некую базовую функцию, которая таким образом формирует другой массив, который, в свою очередь, помещен во вторую функцию, и т. д. Матрешка разрастается, образуя, в конце концов, генерализованный объект, выражающий переработанные свойства исходного массива данных – результат расчетов. Такая запись на языке R есть одновременно и программа, и обозначение массива с результатами обработки. Наша задача состоит в том, чтобы выработать у читателя навык записи своих целей в форме структуры.

Необходимая дробность записи программы создается не сразу, скрипт разрабатывается постепенно – он начинается с формулировки общей задачи, которая дробится и, в конце концов, становится структурированным алгоритмом. Описанный подход в явной форме выражает принцип иерархии («разделяй и властвуй») – один из способов системного мышления: нечто большое и непонятное дробится на более мелкие и более понятные фрагменты. Дробление продолжается до тех пор, пока прежде непо-

нятный объект, сложенный теперь из понятных частей, становится в целом более понятным.

Мы будем составлять программы на R, используя этот принцип. Сначала алгоритм процесса отобразим как две позиции: «дано» – «получить». Затем рассмотрим более глубокую структуру – некие этапы достижения результата. Если потребуется, углубимся на следующий уровень иерархии, и т. д., до тех пор, когда вместо слов можно будет писать операторы и функции R.

Наша цель – обучить программированию в процессе создания программ.

Зачем нужен язык R в биологии

Представим с помощью принципа иерархии место языка R в научной деятельности биолога. На первом уровне определяем общий алгоритм.

Дано: биологический объект (например, конкретная популяция).

Получить: знание о функционировании объекта (например, отчего эта популяция имеет вполне определенную численность?).

Задаём этапность процесса познания и его промежуточные результаты (выделенный шрифт указывает на этапы с использованием R).

Формулировка научной цели
Сбор данных
Организация базы данных
Постановка цели обработки и подбор метода
Организация выборки
Построение диаграммы
Проведение расчетов
Статистический и содержательный выводы

Нас интересуют три этапа (выборка, обработка, диаграмма), которые составляют наш предмет рассмотрения и требуют определения своей структуры. Ведение базы данных также можно организовать средствами R, однако для начинающих ввод информации и исправление ошибок проще выполнять в среде MS Excel или Access. Из этих этапов обработки данных наиболее сложен, конечно, статистический анализ. Его теория и методы обсуждались в наших пособиях, конкретные примеры представлены на стр. 81–106. Далее рассмотрим процессы извлечения выборок из базы данных и построения информативных диаграмм.

Развернутый алгоритм будет включать ряд последовательно реализуемых этапов.

***Дано*:** биологический объект (например, популяция определенного вида)

Формулировка научной цели и программы

Наблюдение за объектом

Накопление информации об объекте (в базе данных)

Формулировка цели обработки (поиск отличий или зависимости)

Извлечение выборки

Дано: база данных

чтение базы данных из файла в массив

формирование выборки из базы по критериям

Дано: массивы базы данных

проверка данных

отбор полей

объединение таблиц

построение фильтра

отбор записей

отбор полей

Получить: массив выборки

***Получить*:** выборка для обработки

Построение диаграммы

***Дано*:** выборка данных

выбор переменных

выбор типа диаграммы

выбор графических средств

разметка и форматирование осей

построение диаграммы

формирование легенды

***Получить*:** диаграмма

Выполнение процедуры статистического анализа

***Дано*:** цель, метод и данные

форматирование выборки под конкретный метод

выполнение статистической процедуры

вывод результатов на экран или в файл

***Получить*:** статистический вывод

Формулировка статистического вывода

Формулировка содержательного вывода

Получить: знание о механизме поддержания численности (публикация).

Три вида программирования

Не вдаваясь в детали, технологии программирования и соответствующие языки можно разделить на три типа: алгоритмические, структурные, объектно-ориентированные.

Программировать на *алгоритмическом языке* проще всего: команды выполняются последовательно, результат выдают в конце. Однако такие программы сложно читать, сложно отлаживать, они уникальны в смысле достижения только одного результата, в них нет библиотеки программ. Таков был первый вариант языка Бейсик. В R есть все возможности и все операторы такого языка, однако «в лоб» программировать алгоритмы на R как-то «не крикет», не эффективно.

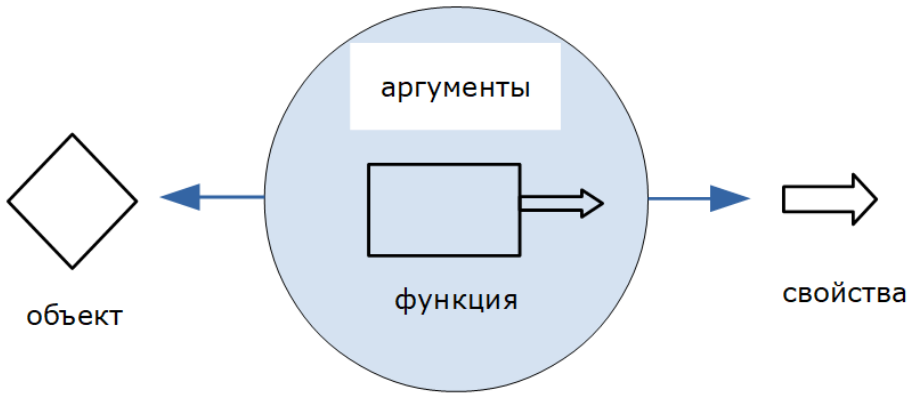
При *структурном программировании* код делят на функциональные блоки, которые могут быть записаны и отлажены отдельно; они предписывают получение какого-то частного результата. Такие программы-блоки становятся универсальными и их можно использовать (включать или подключать) в разных программах, из них можно составлять библиотеки процедур. В ранних языках (Fortran, TurboBasic, Pascal) функцию структурирования выполняли «подпрограммы», «функции», «процедуры» и пр. В среде R у каждого пользователя есть возможность написать свои *функции*, выполняющие определенные повторяющиеся действия.

Объектно-ориентированные языки, пришедшие с технологией MS Windows, обслуживают визуально воспринимаемые *объекты* – окна, формы, кнопки, списки и пр., в которых удобно и необходимо работать мышкой. Эти объекты воспринимают действия пользователя и, применив определенный метод обработки информации, *пользователь активно меняет свойства выбранного объекта* (кликаем крестик – окно закрывается). Таков был Visual Basic как очередной этап развития Бейсика. R – объектно-ориентированный язык, в котором программист: а) создает объекты и б) меняет их свойства.

Компоненты языка R

Как утверждают, в R все – объекты. В первую очередь, векторы, массивы и пр. В дидактических целях R можно воспринимать как совокупность компонентов всего пяти типов. В тексте программы необходимо научиться распознавать *объекты* (*массивы*), *свойства* объектов, *функции*, *аргументы* функций, *операторы*.

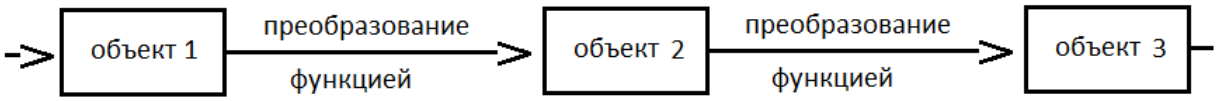
Массивы – это объекты, предназначенные для хранения информации, это области памяти со своими именами. Одномерный массив (вектор) состоит из серии ячеек со значениями. Чтобы создать вектор из четырех ячеек, в которых, например, хранятся числа 5, 6, 7 и 8, используем функцию `c()`: `c(5,6,7,8)`. Этот массив можно переименовать, обозначив буквой, например, x: `x<-c(5,6,7,8)`. В тексте программы *имя*



массива отличается от других компонентов произвольным набором символов; после него могут стоять квадратные скобки с индексом нужного элемента массива: **`x[3]`** (в примере третья по порядку ячейка с числом 7). Имя массива может содержать знак доллара **`$`**, призванного разделить имена основного и подчиненного массива; запись **`mm$spic[3]`** показывает, что вызывается третья ячейка вектора **`spic`**, содержащегося в двумерном массиве **`mm`**. В среде R много классов объектов, но в практическом плане важно только различать массивы класса таблица, список и матрица (см. ниже).

Свойства объектов – это тип данных (числовое, целочисленное, логическое, текстовое, фактор...), размерность (количество и структура ячеек), содержание этих ячеек (значения). В тексте кода свойства либо присваиваются отдельным ячейкам памяти (например, **`x[2]<-23`**), либо задаются как значения аргументов функций **`mean(x, trim = 0.10)`**, либо изменяются функциями.

Функции выполняют обработку, преобразование объектов, в результате которой могут изменить *класс объекта* или изменить *свойства объекта*.



Когда четыре варианты объединяем в одномерный вектор (**`c(5,6,7,8)`**), – это преобразование одного объекта в другой. Когда даем ему имя новое имя **`x`**, – это создание нового объекта. Когда делим вектор на константу, создаем новый объект с изменёнными свойствами.

```
x<-c(5,6,7,8);x
[1] 5 6 7 8
x5<-x/5; x5
[1] 1.0 1.2 1.4 1.6
```

Другая функция (**`mean(x)`**) по-иному формирует новый объект: все значения вектора преобразует в одно среднее, т. е. меняет такие свойства, как длину массива (было че-

тыре ячейки, стала одна) и содержание (вместо 5, 6, 7, 8 стало 6.5), а также тип данных (были целые числа, стало вещественное число).

```
mean(x)
[1] 6.5
```

Несколько записей можно объединить в одну, создав компактный код:

```
mean(c(5,6,7,8)/5)
[1] 1.3
```

В тексте программы функция отличается от иных записей тем, что обязательно снабжается круглыми скобками (в которых приводят аргументы): **`c(5,6,7,8)`**, **`c(5:8)`**, **`mean(x)`**, **`plot(x)`** и т. д.

Аргументы – заключенный в круглые скобки список указаний для функции, что она должна обрабатывать и при каких условиях. Отличить аргументы от других слов можно, во-первых, по тому, что они помещены в круглые скобки функции, во-вторых, что после них ставится знак присвоения «=». После этого знака указывается значение для данного аргумента. Аргументы есть практически у любой функции, как минимум это – имя объекта, с которым функция должна работать; как правило, оно стоит на первом месте. Некоторые функции имеют десятки аргументов. Однако это не осложняет программирование, поскольку язык R очень «дружелюбен» – большинству аргументов еще перед вызовом функции автоматически, «по умолчанию», присвоены некоторые значения. Они описаны в справке к функции и могут быть изменены программистом. Справку вызывают, указав в командной строке имя функции после знака вопроса (пример: **`?mean`**).

О*ператоры* – это тоже функции преобразования данных, но записанные кратко, значками. В их составе процедуры присвоения (**`<-`**, **`=`**) значений массивам (точнее, назначение объектам новых имен), сравнения (**`<`**, **`>`**,**`...`**) данных, арифметические (**`*`**, **`/`**, **`...`**) действия. Они описаны в разделах справки **`?Syntax`**, **`?Arithmetic`**, **`?Logic`**, **`?Comparison`**. Сюда же можно отнести и управляющие функции (**`if()`**, **`for()`**, **`while()...`**), которые задают порядок действий и связывают отдельные функции в программу (см. справку: **`?Control`**).

Центральный объект языка R

Язык R ориентирован на обработку одномерных массивов, векторов. Одномерный массив состоит из серии ячеек со значениями. Массив, состоящий из одной ячейки, рассматривается как вектор с размерностью 1. Двумерный массив следует рассматривать как совокупность вертикально расположенных векторов, приставленных друг к другу сбоку, т. е. в двумерном массиве (таблица) отдельный вектор – это столбец (синонимы – поле, колонка; field, column). Эффективность языка R во многом

определяется тем, что функции обрабатывают векторы целиком. Если в векторе находится 5 элементов, то любое действие, например умножение на скаляр, приводит к появлению вектора из 5 элементов, каждый из которых увеличен в 3 раза.

```
c(2:6)
[1] 2 3 4 5 6
c(2:6)*3
[1]  6 9 12 15 18
```

Имена массивов

Для идентификации массивов данных в R можно (не обязательно) использовать произвольные имена, при создании которых мы рекомендуем придерживаться следующих правил:

1) Нужно избегать давать имена, похожие на служебные слова (NA, T, FAL и др.).
2) R различает прописные и строчные буквы, имена следует давать только строчными латинскими буквами, поскольку многие заглавные и строчные похожи между собой (p и P), на цифры (1 и I), на кириллицу (P и P, c и c), и их легко перепутать, что ведет к ошибкам.
3) Имена могут включать цифры, специальные символы, подчеркивания и точку. Точку следует включать для обозначения значимых промежуточных результатов (xy.reg), но в именах рабочих массивов ее лучше избегать, чтобы не увеличивать размер имени.
4) Длина имени может быть 255 символов, но мы рекомендуем давать ему минимальную длину, чтобы не ошибиться при записи в функции. Лучше давать говорящие имена. Если вы обрабатываете данные по массе зверей (**w**eight of **m**ammals), базовый массив можно назвать **wm**. Если из этого массива извлекаются отдельно самцы (**m**ales), имя этой части данных именуем **mwm**. Если для этого ряда рассчитывается, например, скользящая средняя по пятеркам и в результате обработки получаем массив частных средних, тогда в его имя добавляем **me5mwm**. Такое имя имеет небольшую длину и поддаётся расшифровке (которую все же следует записать в отдельном файле или в комментариях в программе).

Функция как имя объекта

В среде R *запись способа преобразования объекта становится как бы именем преобразованного объекта*; результату преобразования одного объекта в другой не обязательно присваивать новое имя. *В этом блоке будут приведены примеры, которые станут понятными только после прочтения раздела 2; мы рекомендуем перечитывать этот подраздел по мере освоения материала.*

Создадим вектор, состоящий из четырех чисел.

```
c(5:8)
[1] 5 6 7 8
```

С одной стороны, функция `c()` действительно работает, объединяя четыре числа в вектор. С другой стороны, этот вектор получает и свое имя `c(5:8)`. Он имеет свойства одномерного массива с целочисленными значениями с размерностью 4. В этом можно убедиться, рассмотрев его структуру, указав новое имя в функции `str()`.

```
str(c(5:8))
int [1:4] 5 6 7 8
```

Запись `c(5:8)` работает как имя при обращении к отдельной ячейке вектора: указываем индекс в квадратных скобках, например, для третьей ячейки.

```
c(5:8)[3]
[1] 7
```

Запись `c(5:8)` работает как имя при добавлении новых функций: частное от деления вектора на константу обладает теми же свойствами (круглые скобки нужны для арифметической процедуры деления).

```
c(5:8)/5
[1] 1.0 1.2 1.4 1.6
(c(5:8)/5)[3]
[1] 1.4
```

Та же логика применима и к *извлечению из массива нужных значений*. Пусть массив содержит 20 элементов.

```
seq(1:20)
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Тогда из него можно поэтапно извлекать блоки, например, с 2-го до 8-го.

```
seq(1:20)[2:8]
[1] 2 3 4 5 6 7 8
```

Из этого нового объекта с именем `seq(1:20)[2:8]` тем же приемом можно извлечь другие данные, и т. д.

```
seq(1:20)[2:8][2:3]
[1] 3 4
```

Аналогичным способом удобно *извлекать нужные значения из объектов*, в которых статистические функции записывают результаты обработки. Например, функция **lm()** проводит регрессионный анализ зависимости массы (w) от длины тела (lt) полёвок (w=*a*+*b**lt). Она формирует обширный объект класса **Список** (List), в котором, помимо прочей информации, в блок $coefficients помещены оценки коэффициентов регрессии (*a* и *b*) (здесь **mm$w** – масса тела, **mm$lt** – длина тела животных).

```
str(lm(mm$w~mm$lt))
List of 13
 $ coefficients : Named num [1:2] -35.507 0.633
  ..- attr(*, "names")= chr [1:2] "(Intercept)" "mm$lt"
 $ residuals    : Named num [1:53] -0.6 2.99 2.33 -2.13 5.39 ...
  ..- attr(*, "names")= chr [1:53] "1" "2" "3" "4" ...
 $ effects      : Named num [1:53] -158.99 57.69 2.4 -1.84 ...
  ..- attr(*, "names")= chr [1:53] "(Intercept)" "mm$lt" "" ""..
 $ rank         : int 2
```

Эти коэффициенты можно извлечь, если указать имя блока с коэффициентами ($coefficients):

```
lm(mm$w~mm$lt)$coefficients
Coefficients:
(Intercept)        mm$lt
   -35.5071       0.6327
```

Другой вариант вызова отдельного вектора из массива List – двойные квадратные скобки с номером вектора.

```
lm(mm$w~mm$lt)[[1]]
(Intercept)        mm$lt
   -35.5071       0.6327
```

Для вызова отдельного коэффициента указываем его индекс.

```
lm(mm$w~mm$lt)$coefficients[2]
    mm$lt
0.6327023
```

Программа на R – это «матрешка»

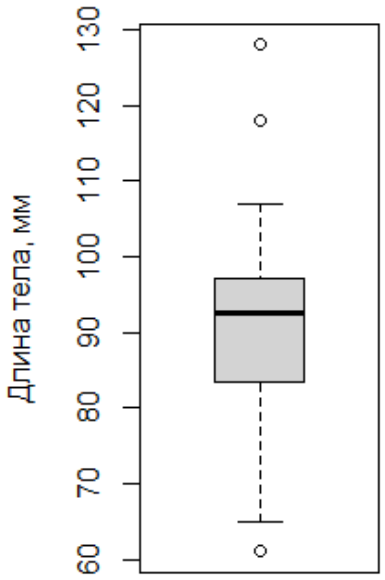
Поскольку и исходное имя массива, и способ его преобразования, и результат этого преобразования записывается как составное имя, *программирование на R можно представить как составление имени нового объекта, а всю программу – как имя искомого массива*. Программа, составленная таким образом, подобна матрёшке, внутрь которой помещается объект, вмещающий в себя другой объект, и т. д. Такой способ

программирования коренным образом отличается от алгоритмического; он гораздо более компактен и «красив». Например, построение графика линии регрессии зависимости массы от длины тела у полевок можно записать «матрешкой» с четырьмя вложениями.

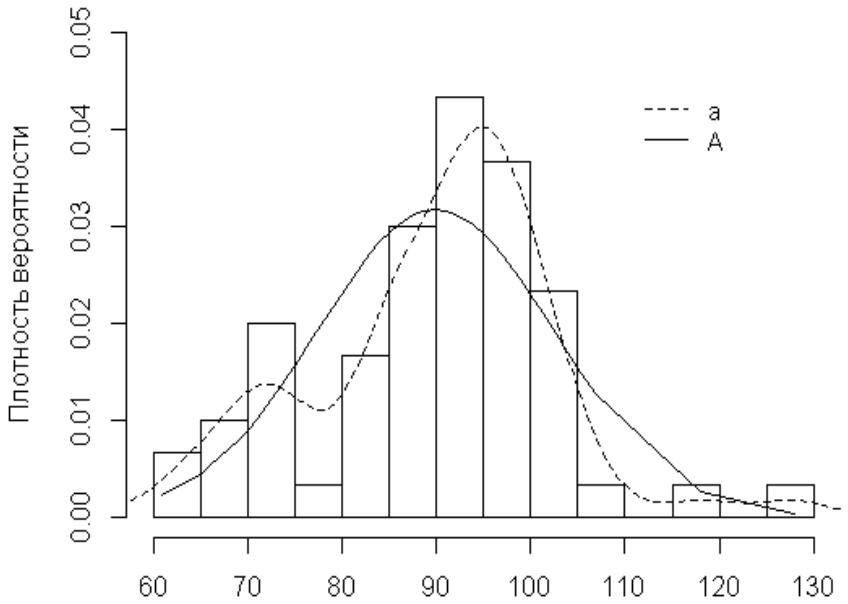
```
plot(mm$lt, predict(lm(mm$w~mm$lt)), type='l').
```

Построение диаграмм

Отображение данных на диаграммах упрощает разведочный анализ, позволяет выявить тренды и зависимости, помогает проиллюстрировать полученный вывод. На практике чаще других строят линейные графики, диаграммы рассеяния и разброса, гистограммы. Помимо базовых функций (**`plot()`**, **`boxplot()`**, **`hist()`**), которые создают поле диаграммы и наносят на него данные, существуют дополнительные функции, добавляющие другие данные (**`line()`**, **`points()`**, **`text()`**), форматирующие оси (**`axis()`**), формирующие легенду (**`legend()`**).

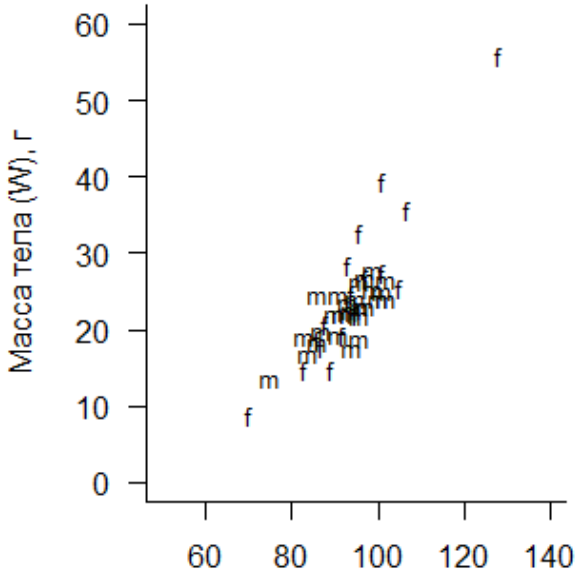


`boxplot(…)`

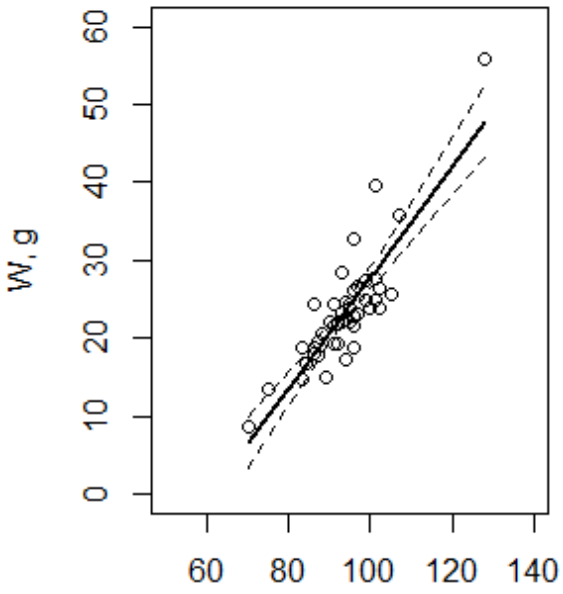


`hist(…); plot(…); legend(…)`

В процессе построения диаграмм для первичной визуализации данных (например, массива **`x`**) записывают базовую функцию с единственным аргументом – именем массива данных, **`hist(x)`**. Значения остальных аргументов программа R задает по умолчанию. Затем рисунок настраивают, добавляя нужные значения аргументам – меняют название и масштаб осей, размерность, способы графического представления, перечисляя соответствующие аргументы через запятую (например, **`hist(x, xlab="length", xlim=c(60, 130))`**). И в конце добавляют дополнительные элементы – оси, легенду и пр.



`plot(…); text(…)`



`plot(…); points(…); lines(…)`

Организация базы данных

На протяжении всей книги практика работы в R будет рассмотрена на примере обработки фрагмента реальной базы данных по мелким млекопитающим. Вначале будут рассмотрены общие принципы ее организации, затем – структура конкретной базы.

Проводя эксперимент или отбирая пробы в полевых условиях, биолог получает разнородную информацию, которую удобнее хранить в разных таблицах базы данных. Зоолог в одну таблицу внесет информацию о характеристиках зарегистрированных животных, во вторую — внесет описание проб (или опытов) – где, когда и в каком объеме проводились наблюдения. Для полных видовых названий следует составить отдельную таблицу, а в основной таблице хранить коды названий, например, однозначные короткие аббревиатуры вроде **sar** для ***S****orex* ***ar****aneus*. Нет никакого смысла включать в базу данных по животным полные видовые названия. Помимо времени, которое просто пропадает, при вводе полных названий неизбежны ошибки. То же можно сказать про описания территорий, где проводятся наблюдения, – для биотопов нужна отдельная таблица, а в таблице для животных достаточно вносить код данного биотопа (bog, meadow и пр.). Получается, что зоологическая база данных содержит четыре таблицы – одна характеризует особей (**Mammalia**), другая – пробы (линии давилок, **Lines**), третья – видовые названия (**Species**), четвертая – описания местообитаний (**Biotop**).

Базы данных содержат одну или несколько таблиц с данными. Двумерные таблицы в среде Excel состоят из *рядов* (строк) и *столбцов* (колонок). В этой электронной таблице соседние ячейки в одном столбце могут содержать данные разных типов.

К двумерным таблицам в среде Access предъявляются более жесткие требования. В одном столбце должны быть данные только одного типа. Ряды данных в них называются *записи*, столбцы называются *полями*. В среде R отдельный вектор (столбец) двумерной таблицы также должен иметь один тип данных. В дальнейшем изложении мы будем пользоваться терминами *поле*, *колонка*, *столбец*, *column*, *field* как синонимами, то же – и для терминов *ряд*, *строка*, *запись*, *row*.

Обычно в одной таблице данных содержится три рода сведений. Во-первых, несколько полей содержит информацию, общую для очень многих записей. Например, для базы по особям животных – это поля год, месяц, номер промыслового орудия, место отбора. Во-вторых, ряд полей несут индивидуальную информацию (отражающую статус особи), которая может быть такой же и для многих других особей, например, пол, вид, зрелость и др. Третья группа – это уникальные характеристиками особей, в числе которых масса, размеры тела, температура, потребление кислорода и пр. Кроме «естественных» полей таблицы данных приходится дополнять полями с расчетными значениями; это могут быть различные индексы (отношения исходных значений для каждой записи) или значения, приведенные к другой единице измерения, и пр. Последняя, пятая, группа – это составные ключи.

Ключом, ключевым полем, называют поле в базе данных, содержащее значения, отличающие одни записи (объекты) от других. В качестве ключей могут выступать любые типы данных и любые поля таблицы – все определяется потребностями исследователей. Комбинируя два или несколько исходных полей, можно создать дополнительное поле – составной ключ. Смысл хранения и накопления информации в электронном виде главным образом состоит в том, чтобы анализировать эти данные на компьютере, в частности, с помощью статистических методов. Для этого из исходной таблицы нужно извлечь выборку, которую может воспринять соответствующая программа статистической обработки. Условия для выборок оформляются в виде *запроса*. *Простой запрос – запись условий отбора данных из одной или нескольких таблиц и одновременно виртуальная новая таблица,* которая может быть сохранена в отдельном массиве или файле. В среде R простые запросы составляются с помощью логических операторов сравнения (`==`, `<`, `>` . . .) и указания индексов полей и записей (номеров столбцов и рядов в квадратных скобках).

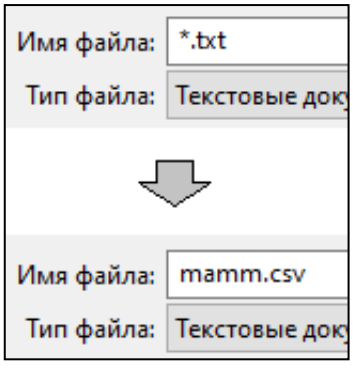
В современной реляционной (relation – отношения) базе данных между всеми таблицами организованы «отношения» – связи по ключевым полям. В каждую пару таблиц, которые нужно связать, вносят одно общее поле, однозначно идентифицирующее записи в этих таблицах. Например, и таблица для животных, и таблица видовых названий содержат поле с кодами видов – cgl, sar, smi и др. С помощью команд управления между этими таблицами устанавливается связь. Это тоже запрос. *Сложный запрос – это и запись условий объединения двух таблиц, и виртуальная новая таблица, состоящая из (частей) исходных таблиц.* Связав таблицы **Mammalia** и **Species**, мы к имеющимся характеристикам каждой особи дописываем еще и их полное имя. Понят-

но, что в запрос можно включать не все поля и записи объединенных таблиц. Для этого в запрос добавляются условия отбора нужной информации. Таким образом составляют одну таблицу (выборку) из двух таблиц. В среде R для связи таблиц служит функция `merge()`.

Для связи двух таблиц необходимо заранее подготовить *общие ключи*, которые будут однозначно идентифицировать записи, относящиеся к одному моменту времени или одному и тому же месту проведения наблюдений. Часто не удается связать такие базы по одному ключу и приходится в каждой из них делать *составной ключ*, который занимает отдельное поле и объединяет в себе несколько полей, идентичных в обеих таблицах. Например, может быть ключ, совмещающий в себе информацию о годе, месяце проведения наблюдений и о номере проб, взятых в этот промежуток времени в этом месте: «год – месяц – место». Такой ключ представлен в нашей базе – это yeseli: year*100000+season*1000+line. Нет препятствий к тому, чтобы рассчитывать составные ключи в среде R. Важно только помнить, что оперативно составленные ключи могут содержать не выявленные ошибки и приводить к неверным результатам. Иногда может быть проще ключи конструировать в среде специальной базы данных (MS Access), поскольку в ней есть средства для проверки и исправления возможных ошибок значений ключа.

Опорные данные

Нами использован фрагмент базы данных по отловам мелких млекопитающих, отловленных в окрестностях стационара КарНЦ «Гомсельга» в 2001–2003 гг. в весенний период в линии 1, 2, 3. Она представлена в 3 таблицах формата csv. Сокращённая таблица **mamm.csv** содержит характеристики каждой отловленной особи; номер особи в журнале (n), вид особи (spic), год (year), сезон отлова (season), пол (sex), зрелость (fer), возраст (age), составной ключ (yeseli), номер линии (line), массу (w), длину тела (lt) и хвоста (lc), длину стопы (lp). Описания линий ловушек содержит вторая таблица **lines.csv**: номер линии в данный сезон (nli), сезон (se), год (ye), дату начала работы (da), число отработанных давилко-суток (traday), тип биотопа (biotop), составной ключ (yeseli). Третья таблица **spic.csv** содержит полную расшифровку видовых названий: код (spic), полное название (names).



Для выполнения предложенных примеров нам понадобятся три файла: **mamm.csv**, **line.csv**, **spec.csv**, сформированные из представленных ниже данных. Каждую таблицу следует скопировать в окно блокнота Windows и сохранить в виде отдельного файла, изменив с клавиатуры расширение *.txt на расширение *.csv в строке Имя файла окна Сохранение – для первой таблицы **mamm.csv** (важно, чтобы в сохраненной таблице не добавилась пустая строка снизу).

Таблица по особям **mamm.csv**

n,spic,year,season,sex,fer,age,yeseli,line,w,lt,lc,lp
976,cgl,2001,6,m,ad,9,200106002,2,24,95,40,16
977,sar,2001,6,f,ad,1,200106002,2,12.4,71,36,12
978,cgl,2001,6,m,sad,2,200106002,2,24.4,91,39,16
979,cgl,2001,6,m,sad,2,200106002,2,25,99,43,17
980,cgl,2001,6,m,ad,9,200106002,2,24.3,86,45,16
981,cgl,2001,6,m,ad,9,200106002,2,23,97,44,17
982,cgl,2001,6,f,ad,9,200106001,1,32.7,96,41,16
983,smi,2001,6,f,ad,1,200106001,1,6.7,61,32,9
984,cgl,2001,6,f,sad,2,200106002,2,18.7,86,40,16
985,cgl,2001,6,f,sad,2,200106002,2,27.6,101,44,16
986,sar,2001,6,m,ad,1,200106002,2,10.4,74,39,12
1489,cgl,2002,5,m,ad,,200205002,2,23.8,102,43,17.7
1494,cgl,2002,5,m,ad,,200205002,2,21.7,96,44,18
1495,cgl,2002,5,f,ad,,200205002,2,22,95,35,17
1496,cgl,2002,5,f,ad,,200205002,2,14.9,89,43,17.8
1497,cgl,2002,5,f,ad,,200205001,1,19.3,92,46,17
1500,cgl,2002,5,m,ad,,200205003,3,21.8,92,40,17.4
1501,cgl,2002,5,m,ad,,200205002,2,24.8,101,41,17.7
1502,cgl,2002,5,f,ad,,200205001,1,26.8,97,37,16.6
1507,sar,2002,6,m,ad,1,200206001,1,10.2,73.800000190734 86,36,12.5
1508,cgl,2002,6,m,ad,,200206002,2,17.4,94,42,17.8
1509,cgl,2002,6,f,ad,,200206001,1,55.8,128,39,19
1510,cgl,2002,6,m,ad,,200206002,2,26.3,102,40,17.8
1511,cgl,2002,6,m,ad,,200206002,2,22.4,94,39,16.5
1512,cgl,2002,6,m,ad,,200206003,3,23.9,100,40,18.5
1513,sar,2002,6,,ad,1,200206001,1,11,73,40,12
1514,cgl,2002,6,f,ad,,200206003,3,24.7,94,43,18
1515,cgl,2002,6,f,ad,,200206001,1,35.8,107,33,18.9
1516,cgl,2002,6,f,ad,,200206001,1,39.5,101,33,21
1517,sbe,2002,6,f,ad,,200206001,1,10.5,69,86,17.5
1518,mag,2002,6,f,ad,,200206002,2,42.9,118,34,19
1519,cgl,2002,6,m,ad,,200206002,2,18.7,96,37,17
1520,sbe,2002,6,f,ad,,200206001,1,8.6,66,86,16
1521,cgl,2002,6,m,ad,,200206003,3,22.9,96,38,18
1522,sbe,2002,6,f,ad,,200206001,1,12.9,71,94,17
1523,cgl,2002,6,f,ad,,200206002,2,20.7,88,41,17
1524,cgl,2002,6,m,sad,,200206001,1,26.2,96,26,19
1871,cgl,2003,6,m,ad,,200306002,2,27.5,99,45,19
1872,cgl,2003,6,m,ad,,200306002,2,19.6,87,42,18
1873,cgl,2003,6,f,ad,,200306002,2,17.8,87,38,17.5
1874,cgl,2003,6,m,,,200306002,2,23.4,93,43,17.8
1875,cgl,2003,6,m,,,200306002,2,22,90,43,16.5

1876,cgl,2003,6,f,,,200306001,1,28.5,93,42,18
1877,cgl,2003,6,m,,,200306003,3,13.4,NA,NA,NA
1878,cgl,2003,6,m,,,200306001,1,23.4,NA,NA,NA
1879,cgl,2003,6,m,,,200306003,3,13.4,75,32,17
1880,cgl,2003,6,m,,,200306001,1,16.7,84,41,18
1882,cgl,2003,6,m,ad,,200306002,2,18.1,86,36,18
1883,cgl,2003,6,f,ad,,200306001,1,NA,101,50,17.9
1884,cgl,2003,6,m,ad,,200306002,2,26.6,97,40,17.9
1885,cgl,2003,6,f,ad,,200306002,2,NA,88,36,17
1886,cgl,2003,6,,,,200306001,1,NA,84,42,17.1
1887,cgl,2003,6,m,ad,,200306001,1,18.9,83,37,17.1
1888,cgl,2003,6,f,ad,,200306003,3,NA,95,52,17.1
1889,cgl,2003,6,m,ad,,200306002,2,22.1,93,41,17.2
1890,cgl,2003,6,f,juv,,200306003,3,NA,77,34,16
1891,cgl,2003,6,f,juv,,200306001,1,NA,83,37,17
1892,cgl,2003,6,,juv,,200306001,1,NA,65,30,16.2
1901,cgl,2003,6,f,ad,,200306002,2,25.6,105,50,18
1902,cgl,2003,6,f,ad,,200306002,2,14.7,83,38,17.5
1903,cgl,2003,6,m,ad,,200306003,3,19.2,91,43,18.5
1904,cgl,2003,6,f,juv,,200306001,1,8.7,70,31,15.5

Таблица по объемам отловов в линии **line.csv**

nli,se,ye,da,traday,biotop,yeseli
1,6,2001,19,150,meadow,200106001
2,6,2001,19,150,conifer,200106002
3,6,2001,19,150,mixed,200106003
1,5,2002,4,150,meadow,200205001
2,5,2002,4,150,conifer,200205002
3,5,2002,4,150,mixed,200205003
1,6,2002,24,150,meadow,200206001
2,6,2002,24,150,conifer,200206002
3,6,2002,24,150,mixed,200206003
1,6,2003,19,150,meadow,200306001
2,6,2003,19,150,conifer,200306002
3,6,2003,19,150,mixed,200306003

Таблица по видам **spec.csv**

spic,names
sar,Sorex araneus
sis,Sorex isodon
sca,Sorex caecutiens
smi,Sorex minutus
sms,Sorex minutissimus

nfo,Neomys fodiens
teu,Talpa ueropea
msh,Myopus schisticolor
mmi,Mus minutus
mmu,Mus musculus
aag,Apodemus agrarius
rno,Rattus norvegicus
ate,Arvicola terrestris
mar,Microtus arvallis
moe,Microtus oeconomus
mag,Micritus agrestis
cgl,Clethrionomys glareolis
cru,Clethrionomys rutilus
sbe,Sicista betulina

ПРАКТИКА ПРОГРАММИРОВАНИЯ

Эта глава предназначена для предметного выполнения читателем предлагаемых примеров и заданий в среде R. Запустите R и повторяйте наши примеры на компьютере.

Файл установки программы R находится на сайте https://cran.r-project.org/bin/windows/base/. Запуск инсталлированной программы R – двойной клик на иконке. Выход из программы – команда главного меню Файл \ Выйти или команда консоли `q()`.

После установки программы в среде R доступно некоторое число функций из нескольких базовых пакетов (Base, Stats, Tools, Utils и др.) При необходимости другие пакеты можно загрузить из интернета (команда меню Пакеты \ Установить пакет(ы)) и включить их (Пакеты \ Включить пакет). Для поиска пакетов, содержащих нужную математическую процедуру, служит пакет **sos**, который нужно загрузить и включить. Поиск пакетов выполняет функция `findFn(string = {"нужная процедура"}, maxPages = ?)` (см. подробнее: https://r-analytics.blogspot.com/2012/10/sos-r.html).

Получить справку по любой функции (из подключенного пакета) можно, открыв меню Справка \ Функции R (текст)… или с консоли, введя `?` или `help` и имя функции, например: `?mean` или `help("mean")`. Описание полезных источников дано на стр. 107–111.

Для выполнения отдельных заданий средствами Windows организуйте отдельную папку для сохранения программ и данных (у нас – C:\ r \ examples); имена папок и файлов обязательно должны быть *на латинице*. Подключите R к этой папке с помощью команд Главного меню: Файл \ Изменить папку. То же можно сделать командой с консоли: `setwd("C:/r/examples")` (для разделения подкаталогов в R используется слэш "/", знак деления, а не бэкслэш "\", как в Проводнике Windows). Теперь R будет работать с данными и программами, помещенными только в эту папку.

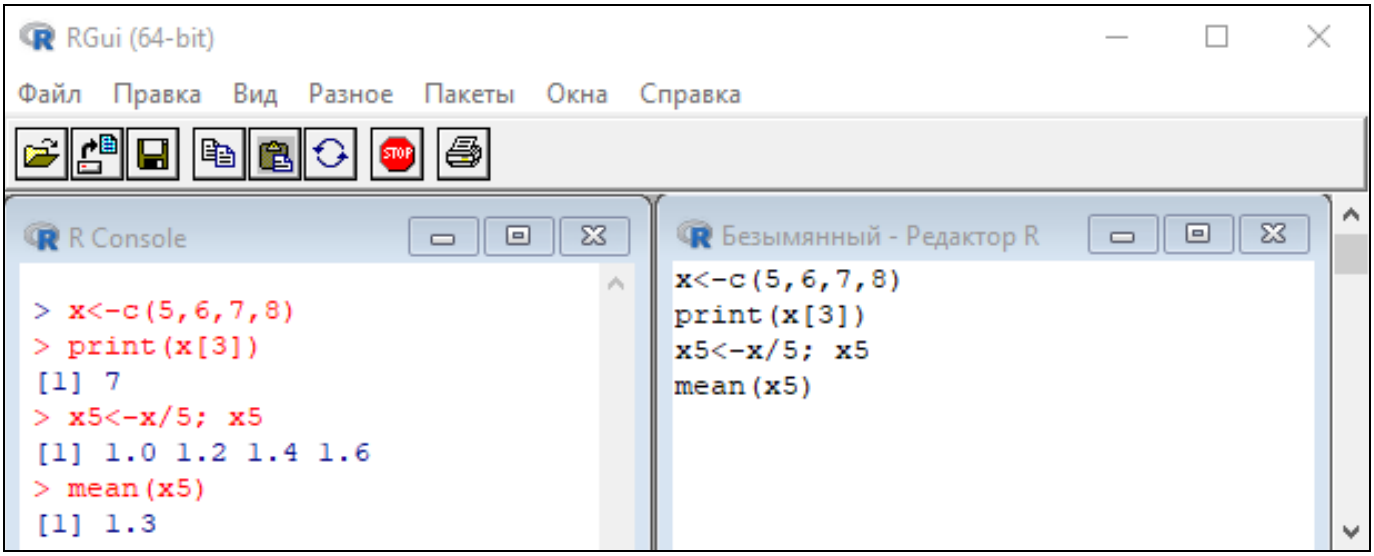
Консоль, скрипт, график

В среде R можно работать в нескольких окнах. Размер окон регулируется мышкой и командами Главного меню Окно (рекомендуем Окно \ Сверху вниз). Главные окна – Консоль (R Console), Скрипт (…– Редактор R) и График (R Graphics).

Окно консоли появляется сразу после запуска R. Окно консоли очищается сочетанием клавиш Ctrl+L или командой главного меню Правка \ Очистить консоль. В окно консоли после приглашения ">" можно ввести любую команду на языке R (красный цвет шрифта), нажать клавишу Enter и тем самым выполнить ее. Результаты выполнения команд также появляются в окне консоли (синий цвет шрифта) или в окне диаграмм (R Graphics). Если команда написана с нарушением синтаксиса или структуры

данных, появится запись, сообщающая об ошибке (синим шрифтом). Если появились какие-то проблемы с выполнением команды и приглашение ">" не появляется, следует нажать клавиши Esc и Enter и управление вернется. Команды в консоль можно ввести разными способами: (а) с клавиатуры, (б) из буфера обмена (Ctrl+V), (в) из списка ранее введённых команд, накопленных в стеке (стрелки вверх-вниз), (г) из окна скрипта (Ctrl+R).

Во всех примерах ниже команда будет записана без значка приглашения – как в скрипте, в том числе для того, чтобы эту команду было удобно скопировать, вставить в свой скрипт и выполнить для тренировки.



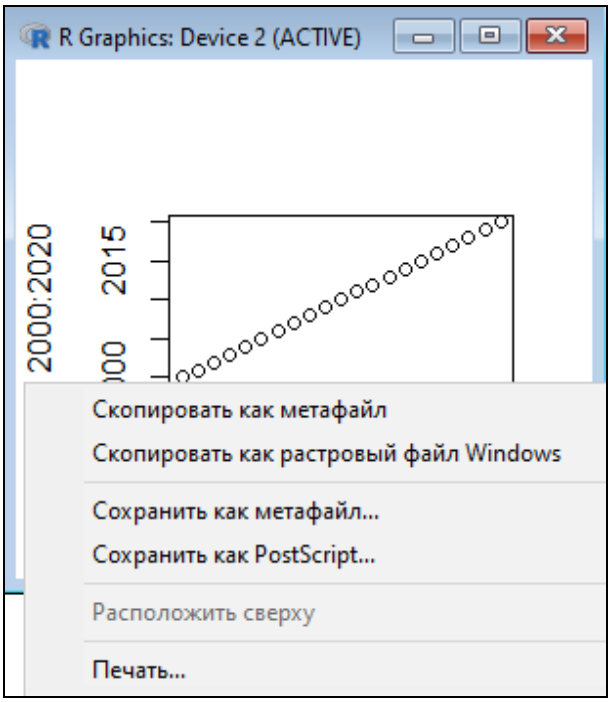
Окно скрипта появляется, если дать команду Файл \ Новый скрипт или Файл \ Открыть скрипт. Оно предназначено для записи и последовательного выполнения нескольких команд (это и есть программа, код, скрипт). По функционалу это простой текстовый редактор, аналогичный Блокноту среды MS Windows, в который можно записывать и редактировать в нем любые символы, используя простые команды, в том числе с помощью буфера обмена (копировать Ctrl+C, вставлять Ctrl+V). Отличие от Блокнота состоит в том, что из среды окна Скрипт можно передать в консоль и выполнить любую команду, записанную на языке R. Для запуска можно пользоваться командами Главного меню (Правка \Запустить строку или блок и Правка \ Запустить все), контекстным меню (клик на строке правой кнопкой мыши), но мы рекомендуем использовать сочетание клавиш Ctrl+R. Так можно запустить разные по размеру блоки команд скрипта:

- если установить курсор в строку и нажать сочетание клавиш Ctrl+R, то выполнятся все команды только данной строки;
- если предварительно мышкой выделить часть строки (например, имя переменной) или несколько строк, сочетание клавиш Ctrl+R выполнит только вы-

деленную команду (так можно сделать вывод содержимого массива на консоль);

- если выделить все команды (Ctrl+A), то сочетание клавиш Ctrl+R выполнит всю программу.

Каждая запущенная из скрипта *команда тут же отображается в окне скрипта* (красным шрифтом). Ниже нее помещается результат выполнения данной команды (синим шрифтом). Если команда записана неправильно, после нее появляется комментарий об ошибке (синий шрифт). В наших примерах команды выделены жирным шрифтом, результаты – простым.



Окно диаграммы открывают функции, создающие диаграммы – **`plot()`**, **`boxplot()`**, **`mathplot()`**, **`hist()`** и др. Диаграмма строится из заранее подготовленных данных, включенных в список аргументов функций. Размер окна диаграммы регулируется мышкой. Правый клик на окне диаграммы открывает диалоговое окно, позволяющее скопировать диаграмму в буфер обмена, вставить в любой растровый редактор и сохранить как растровый файл Word.

Создание объекта (массива)

Массивы содержат информацию, подлежащую обработке; функции R обрабатывают массивы, получая из них другие массивы. Для разных целей в среде R можно организовать массивы-объекты, обладающие различающимися свойствами и разной размерностью. Мы будем использовать два *класса объектов-массивов* (вектор как одномерный массив и таблицу как двумерный массив) и пять *типов данных*: целые, десятичные, текстовые, факторы, логические.

Создание одномерных массивов-векторов

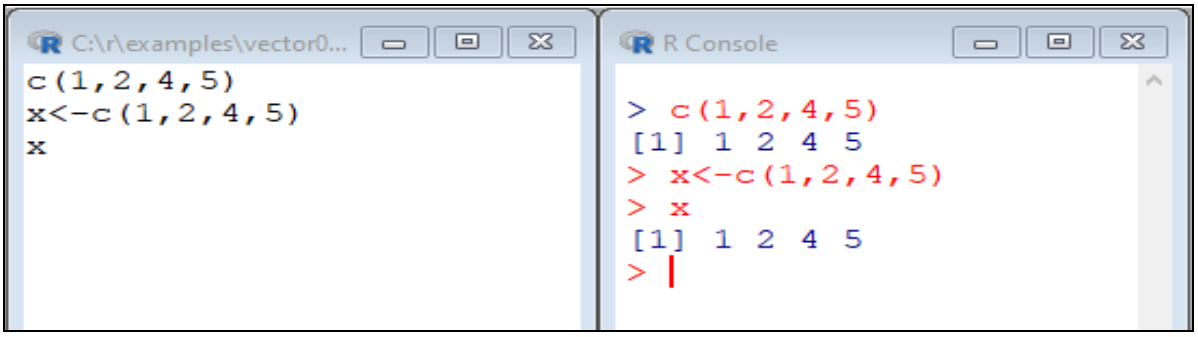
Язык R ориентирован на обработку векторов (одномерных массивов) – наборов ячеек памяти, в которых хранятся значения. Процедуры перебора значений встроены в команды обращения к данным и осуществляются «по умолчанию». Для создания вектора используют функцию **`c()`** (от **c**oncatenation – сцепление), которая сцепляет в массив значения, перечисленные в скобках. Нужно запомнить, что в R для всех функций *аргументы всегда перечисляют через запятую* (например, команда **`c(1,2,4,5)`** создает массив из 4 значений), команду из скрипта запускают сочетанием клавиш Ctrl+R или через меню Правка.

При выводе содержимого массива на консоль в квадратных скобках указывается номер первого элемента в данной строке, это важная информация, если массив длинный и выводится несколько строк.

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

Отдельные векторы можно назвать кратким именем, используя латинские буквы, цифры, символы точки и подчеркивания. Для присвоения массивам оригинальных имен используют оператор присваивания "**<-**": записываем код **`x<-c(1,2,4,5)`** в скрипт, помещаем курсор в эту строку и даем команду Ctrl+R. Для просмотра содержимого массива в окне консоли в окне скрипта следует набрать его имя и дать команду Ctrl+R. В итоге на консоль выводится

```
x<-c(1,2,4,5)
x
[1] 1 2 4 5
```



Структура вывода будет определяться структурой данных, но ее можно отрегулировать, использовав команду **`print()`**, в которой нужно указать число значащих цифр. Через точку с запятой можно ввести несколько команд.

```
x<-c(1,2.45673,4,6); x
[1] 1.00000 2.45673 4.00000 6.00000
```

```
print(x,2)
[1] 1.0 2.5 4.0 6.0
```

Используя внешние скобки (как сокращение функции **print()**), можно увидеть результат выполнения команд (здесь в старый массив добавили два новых элемента).

```
(x<-c(x,5,78))
[1] 1.00000 2.45673 3.45600 6.00000 5.00000 78.00000
```

Определение вектора как последовательности

Одномерный массив как последовательность можно задать с помощью нескольких функций – **seq()**, **rep()**, **sample()**, **runif()**, **rev()**. Функцию **seq()** используют, если нужно сформировать ряд чисел; по умолчанию шаг приращения равен **by=1**.

```
seq(1,15)
[1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
seq(1,15,by=1.5)
 [1]  1.0  2.5  4.0  5.5  7.0  8.5 10.0 11.5 13.0 14.5
```

Функция **rev()** создает обратную последовательность.

```
rev(seq(1,15,by=3))
[1] 13 10  7  4  1
```

Сформировать вектор из чисел в диапазоне от 50 до –10.5 длиной 5 элементов:

```
seq(50,-10.6,length.out=5)
[1]  50.00  34.85  19.70   4.55 -10.60
```

Отдельные фрагменты вектора извлекаются из него с использованием индекса значения в квадратных скобках.

```
seq(1:20)[2:8]
[1] 2 3 4 5 6 7 8
```

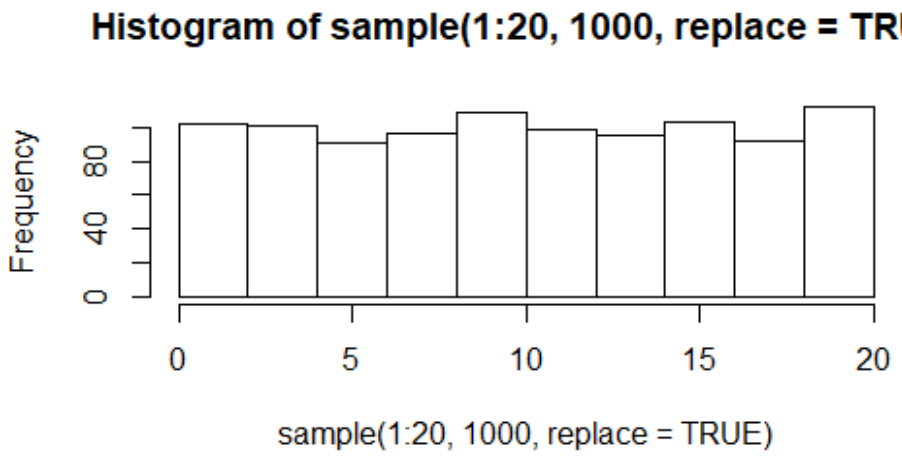
Функция **rep()** служит для повторного воспроизводства чисел.

```
rep(1.4,7)
[1] 1.4 1.4 1.4 1.4 1.4 1.4 1.4
```

Функцию **sample()** используют для извлечения целочисленных равномерно распределенных случайных чисел (в данном случае три числа из диапазона от 1 до 50):

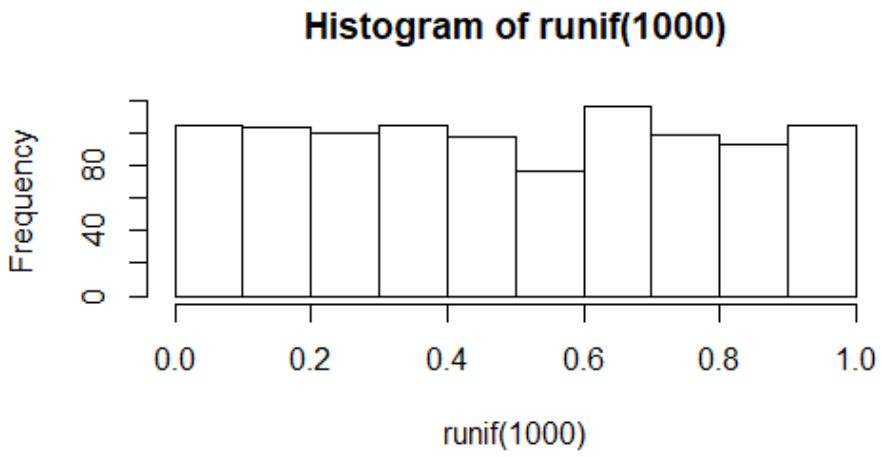
```
sample(1:50,3)
[1] 30 10 23
```

```
sample(1:50,3)
[1] 29 5 25
sample(1:50,3)
[1] 15 34 48
hist(sample(1:20,1000,replace=TRUE))
```



Функция **runif()** служит для извлечения дробных равномерно распределенных случайных чисел (здесь три числа из диапазона от 1 до 50):

```
runif(3,1,50)
[1] 42.68965 31.75496 15.70371
runif(3,1,50)
[1] 33.33669 25.69496 10.52617
runif(3,1,50)
[1] 36.137035 33.866480  9.081742
hist(runif(1000))
```



Формирование одномерного массива при объединении нескольких векторов выполняется также функцией **c()**:

```
c(sample(1:50,3),rep(1.25,3),seq(1,9,3))
[1] 12.00 11.00 44.00  1.25  1.25  1.25  1.00  4.00  7.00
```

Довольно часто при обработке отдельных выборок возникает задача «вытащить» номера строк таблицы, прочитанной из файла. По умолчанию имена строк имеют текстовый формат, но выглядят как числа. Сначала берем *имена* строк, затем преобразуем их в *номера* строк.

```
mm<-read.csv("mamm.csv")
as.integer(rownames(mm))
[1] 1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
[23] 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
[45] 45 4647 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62
```

Вызов значений из вектора

Для обращения к отдельной ячейке массива нужно указать ее индекс (номер) в квадратных скобках. Номер элемента можно хранить в отдельной ячейке и вызывать нужное значение с помощью косвенной ссылки.

```
x<-c(1,2.45673,4,6,5,78)
x[3]<-3.456
a<-5; x[a]<-4560; x
[1] 1.00000 2.45673 3.45600 6.00000 4560.00000 78.00000
```

Обратиться к блоку ячеек можно с помощью двоеточия (в примере создан новый массив из трех элементов старого).

```
x[1:3]
[1] 1.00000 2.45673 3.45600
```

Чтобы *обратиться к ячейкам с помощью списка индексов*, нужно сначала создать его с помощью функции сцепления (запись **c(1,3)** содержит указание на первый и третий элементы массива).

```
x[c(1,3)]
[1] 1.000 3.45600
```

Математические операции и функции (*****, **/**, **+**, **–**, **^**, **sin**, **log**) применяются последовательно ко всем элементам указанного массива.

Создание двумерных массивов

Создание двумерного массива из одного вектора

Один из вариантов формирования двумерных массивов – это создание одномерного массива и его последующее «нарезание» на фрагменты, которые приставляются друг к другу слева направо и образуют двумерный массив. Процедура нарезания выполняется функцией **`array()`**, главными аргументами которой являются исходный вектор (**`data`**) и размерность (**`dim`** – число строк и число столбцов) будущего массива; произведение числа строк на число столбцов должно быть равно длине указанного одномерного массива. Создается объект класса *матрица*.

```
array(data=1:21,dim=c(7,3))
     [,1] [,2] [,3]
[1,]    1    8   15
[2,]    2    9   16
[3,]    3   10   17
[4,]    4   11   18
[5,]    5   12   19
[6,]    6   13   20
[7,]    7   14   21
```

Создание таблицы из нескольких векторов

Векторы с разным типом данных объединяются функцией **`data.frame()`**. При этом формируется массив класса *таблица*. Единственное условие для его создания – одинаковая длина объединяемых векторов. В новом массиве каждый вектор становится отдельным столбцом (полем, колонкой – field, column), а их элементы с одинаковыми индексами образуют ряды (строки, записи – row).

```
data.frame(c('a','b','c'),rep(1.25,3),seq(1,9,3))
  c..a....b....c.. rep.1.25..3. seq.1..9..3.
1                a         1.25            1
2                b         1.25            4
3                c         1.25            7
```

Для лаконичности можно сразу задать имена новым векторам

```
data.frame(nam=c('a','b','c'),x=rep(1.25,3),y=seq(1,9,3))
  nam    x y
1   a 1.25 1
2   b 1.25 4
3   c 1.25 7
```

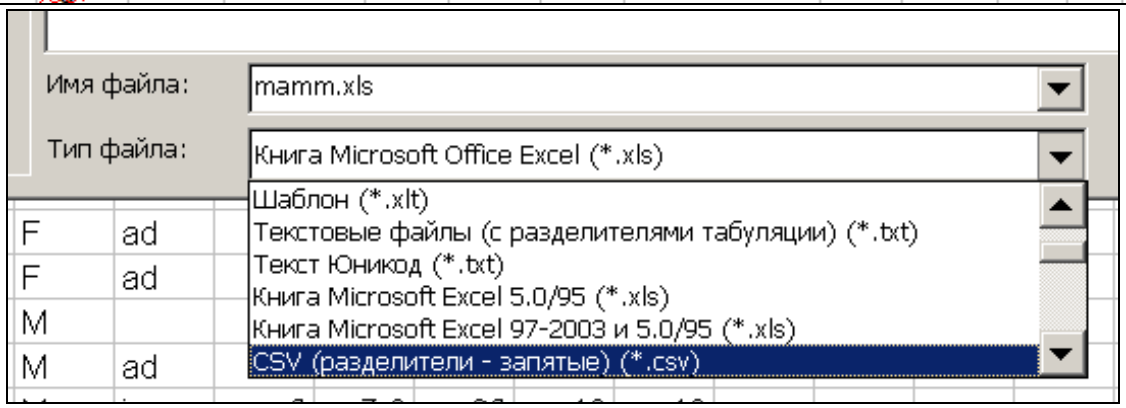
Помимо функции **`data.frame()`** могут быть полезны функции **`array()`**, **`cbind()`**, **`rbind()`**, но они формируют массивы класса *матрица*. Этот формат часто не подходит для расчетов и его приходится превращать в *таблицу* с помощью той же **`data.frame()`**. Есть и более развитые классы таблиц данных – **`tibble()`** и **`data.table()`**.

Создание таблицы при чтении файла

В нашей практике – это самый распространенный метод задания размера и свойств массива. При чтении файла R самостоятельно определяет типы данных и устанавливает размерность массива, равную объему данных в файле. Обычно база данных биологической информации включает несколько десятков значений, например 30 полей и 2000 записей. Для передачи такого небольшого объема данных не нужны специальные способы, вполне достаточно файлов формата csv (comma separated values). В файле каждое значение отделено от других запятой, каждая запись (строка) несет индивидуальный номер, каждое поле (столбец) имеет индивидуальное имя. Этот формат легко экспортируется из среды Excel и Access, может быть отредактирован в Блокноте Windows, однозначно импортируется в среду R.

Наша база данных по полевкам (стр. 18–20) сформирована в среде электронной таблицы (mamm.xls), сохранена в папке examples (C:\r\examples) и экспортирована в файл mamm.csv.

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	n	spic	year	season	sex	fer	age	yeseli	line	w	lt	lc	lp
2	976	cgl	2001	6	m	ad	9	200106002	2	24	95	40	16
3	977	sar	2001	6	f	ad	1	200106002	2	12.4	71	36	12
4	978	cgl	2001	6	m	sad	2	200106002	2	24.4	91	39	16
5	979	cgl	2001	6	m	sad	2	200106002	2	25	99	43	17
6	980	cgl	2001	6	m	ad	9	200106002	2	24.3	86	45	16
7	981	cgl	2001	6	m	ad	9	200106002	2	23	97	44	17
8	982	cgl	2001	6	f	ad	9	200106001	1	32.7	96	41	16



Функция **`read.csv("имя_файла.csv")`** формирует двумерный массив класса *таблица* (**`data.frame`**), объединяющий 13 векторов разных типов длиной по 62 ячейки.

```
setwd("C:/r/examples") # назначение папки с файлом данных

mm<-read.csv("mamm.csv") # чтение файла
head(mm,3) # вывод 3 верхних строк массива
    n spic year season sex fer age    yeseli line    w lt lc lp
1 976  cgl 2001      6   m  ad   9 200106002    2 24.0 95 40 16
2 977  sar 2001      6   f  ad   1 200106002    2 12.4 71 36 12
3 978  cgl 2001      6   m sad   2 200106002    2 24.4 91 39 16
```

```
nrow(mm) # подсчет числа строк
[1] 62
```

Вызов значений из таблицы

Чтобы использовать данные массива в расчетах, нужно *обратиться* к нему по имени. Например, чтобы вывести содержимое массива в окно консоли, нужно ввести имя (**mm**) или включить его в список аргументов функции **print(mm)**. Однако эти функции выведут весь массив данных, что нужно далеко не всегда. Увидеть часть массива можно с помощью команд **head()**, **tail()** или указывая индексы строк.

```
tail(mm,2)
 n spic year season sex fer age yeseli line w lt lc lp
61 1903 cgl 2003  6 m ad NA 200306003 3 19.2 91 43 18.5
62 1904 cgl 2003  6 f juv NA 200306001 1  8.7 70 31 15.5
```

```
mm[15:16,]
 num spic year season sex fer age yeseli line w lt lc lp
15 1496  cgl 2002 5 f ad NA 200205002 2 14.9 89 43 17.8
16 1497  cgl 2002 5 f ad NA 200205001 1 19.3 92 46 17.0
```

Обратиться к *отдельному вектору из двумерного массива* можно, используя индекс поля, имя поля в кавычках или *составное имя (*на первом месте стоит имя общего массива, затем знак доллара и имя столбца, например **mm$sex**):

```
mm[,7]
[1] 9 1 2 2 9 9 9 1 2 2 1 NA NA NA NA NA NA NA NA 1 NA NA NA
[24] NA NA 1 NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
[51] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
```

```
mm[5,'spic']
[1] cgl
Levels: cgl mag sar sbe smi
```

```
mm$sex[15:20]
[1] f f m m f m
Levels: f m
```

Для вызова данных из массивов класса Список (List) адрес значений также оказывается составным. В списке данные разных типов хранятся на двух уровнях. Значения одного типа организованы в векторы со своими именами, в пределах вектора значения проиндексированы. Для извлечения из массива отдельного вектора нужно назвать его имя или индекс ([[номер по порядку в массиве]]), а для извлечения отдельного значения – еще и его индекс ([номер по порядку в векторе]). Например, коэффициенты регрессии в массиве результатов регрессионного анализа находятся в первом векторе с именем `$coefficients`.

```
str(lm(mm$w~mm$lt))
List of 12
 $ coefficients: Named num [1:2] -18.032 0.417
  ..- attr(*, "names")= chr [1:2] "(Intercept)" "mm$lt"
 $ residuals: Named num [1:7] 0.559 1.745 -0.775  …
```

Извлекаем первый вектор:

```
lm(mm$w~mm$lt)$coefficients
(Intercept)       mm$lt
-35.5070769   0.6327023
```

или

```
lm(mm$w~mm$lt)[[1]]
(Intercept)       mm$lt
-35.5070769   0.6327023
```

и значение коэффициента регрессии

```
lm(mm$w~mm$lt)$coefficients[2]
    mm$lt
0.6327023
```

или

```
lm(mm$w~mm$lt)[[1]][2]
    mm$lt
0.6327023
```

Типы данных

Программа должна знать типы данных, поскольку способы их обработки и объемы для хранения в памяти существенно отличаются. Неправильное определение типа данных ведет либо к ошибке в расчетах, либо к снижению возможностей обработки. В дальнейшем мы будем использовать в примерах 5 типов данных: `int` (integer) – целое число, `num` (numerical) – десятичное, или вещественное, число, *разделенное точкой на целую и дробную части*, `char` (character) – любой текст, заключенный в кавычки, `factor` – структурированный текстовый массив, `logi` (logical) – две логические константы `TRUE` (истина) и `FALSE` (ложь).

Тип данных задается *одинаковым для всех значений отдельного вектора*; один столбец не может хранить данные разных типов, как в таблице MS Excel. Если в файле встречаются данные разных типов, R выбирает тот, который не допустит потери информации. Например, если среди целочисленных значений вектора попадется число с точкой, R всему вектору назначит тип `num` (десятичное число). Если же попадется

число с запятой (советский стиль числа), то вектор получит тип Factor (текстовый массив), что осложнит обработку. По этой причине перед экспортом данных из среды MS Excel необходимо тщательно проверить совпадение типов значений в столбцах. Пропуски в числовых полях данных R воспринимает как константы NA (**N**ot **A**vailable), показывающие, что значения не определены. Пропуски в данных типа фактор будут отображаться как текстовые пробелы.

Типы данных можно определить для всех векторов двумерного массива с помощью функции **str()**.

```
str(mm)
'data.frame':   62 obs. of  13 variables:
 $ n    : int  976 977 978 979 980 981 982 983 984 985 ...
 $ spic : Factor w/ 5 levels "cgl","mag","sar",..: 1 3 1 1  ...
 $ year : int  2001 2001 2001 2001 2001 2001 2001  ...
 $ season: int  6 6 6 6 6 6 6 6 6 6 ...
 $ sex   : Factor w/ 3 levels "","f","m": 3 2 3 3 3 3 ...
 $ fer   : Factor w/ 4 levels "","ad","juv",..: 2 2 4 4 2 2 ...
 $ age   : int  9 1 2 2 9 9 9 1 2 2 ...
 $ yeseli: int  200106002 200106002 200106002  ...
 $ line  : int  2 2 2 2 2 2 1 1 2 2 ...
 $ w     : num  24 12.4 24.4 25 24.3 23 32.7 6.7 18.7 27.6 ...
 $ lt    : num  95 71 91 99 86 97 96 61 86 101 ...
 $ lc    : int  40 36 39 43 45 44 41 32 40 44 ...
 $ lp    : num  16 12 16 17 16 17 16 9 16 16 ...
```

Здесь *целые числа* int имеют 7 векторов (n, year,...), *десятичные числа* num – три вектора с промерами (w, lt,...). *Текстовые значения* заключены в кавычки – "cgl", "mag", "f", "m" и т. д. Для повторяющихся текстовых констант в R задается тип Factor, который определен для трех векторов (spic, sex, fer). Такой одномерный массив состоит из двух блоков – текстового и целочисленного. Первый содержит упорядоченный по алфавиту список *текстовых значений* (char). Каждое значение имеет свой номер позиций: в поле sex позиция 1 отведена пробелу ("" – пол зверька не был определен), у самок ("f") позиция 2, у самцов ("m") позиция 3. Второй блок, целочисленный вектор, содержит множество значений, соответствующих каждой строке (особи), – номера позиций текстовых значений из первого списка. Для признаков пола последовательность 3 2 3 3 3 3 … соответствует перечню самок и самцов, содержащихся в таблице: m f m m m m.

Для определения типа данных отдельного вектора используется функция **is.…()**.

```
is.factor(mm$sex)
[1] TRUE

is.integer(mm$num)
[1] TRUE

is.numeric(mm$num)
[1] FALSE
```

Логические значения TRUE и FALSE не фигурируют в нашей таблице, но являются результатом логических операций с данными, например, при проверке выполнения условий. По результатам проверки, имеют ли все векторы массива mm тип int (**is.integer**), формируется массив логических значений:

```
lmm <- sapply(mm,is.integer); lmm
 n num spic mon year sex fer line w lt lc lp
TRUE TRUE FALSE TRUE TRUE FALSE FALSE TRUE FALSE TRUE TRUE TRUE
```

```
is.logical(lmm)
[1] TRUE
```

Поскольку константам FALSE и TRUE соответствуют значения 0 и 1, они легко включаются в арифметические операции, при работе с индексами.

```
sum(lmm)
[1] 8
```

```
lmm/1
 n  num spic  mon year  sex  fer  line   w   lt   lc   lp
 1    1    0    1    1    0    0    1    0    1    1    1
```

```
lmm*50
 n  num spic  mon year  sex  fer line    w   lt   lc   lp
 50  50    0   50   50    0    0   50    0   50   50   50
```

Изменение типа данных

Для изменения типа данных служат функции группы **as.…()**. Необходимо учитывать, что превращение одного объекта в другой сказывается на содержании данных, которые могут утратить часть информации или приобрести нежелательную. Например, преобразование чисел с плавающей точкой в целые приводит к простому отбрасыванию дробной части.

```
mm$w[5:10]
[1] 24.3 23.0 32.7  6.7 18.7 27.6
```

```
as.integer(mm$w)[5:10]
[1] 24 23 32  6 18 27
```

Преобразование фактора в целое «вытаскивает» блок позиций.

```
mm$spic [10:15]
[1] cgl sar cgl cgl cgl cgl
Levels: cgl mag sar sbe smi
```

```
as.integer(mm$spic)[10:15]
[1] 1 3 1 1 1 1
```

Преобразование вектора логических констант дает серию нулей и единиц.

```
lmm[3:7]
  year season    sex    fer    age
  TRUE   TRUE  FALSE  FALSE   TRUE
```

```
as.integr(lmm[3:7])
[1] 1 1 0 0 1
```

Для преобразования всех векторов двумерного массива нужно привлечь функцию обработки векторов **sapply()**.

```
sapply(mm,as.integer)[1:5,]
       n spic year season sex fer age    yeseli line  w lt lc lp
[1,] 976    1 2001      6   3   2   9 200106002    2 24 95 40 16
[2,] 977    3 2001      6   2   2   1 200106002    2 12 71 36 12
[3,] 978    1 2001      6   3   4   2 200106002    2 24 91 39 16
[4,] 979    1 2001      6   3   4   2 200106002    2 25 99 43 17
[5,] 980    1 2001      6   3   2   9 200106002    2 24 86 45 16
```

```
sapply(mm,as.character)[50:53,1:7]
     n      spic  year   season sex fer  age
[1,] "1884" "cgl" "2003" "6"    "m" "ad" NA
[2,] "1885" "cgl" "2003" "6"    "f" "ad" NA
[3,] "1886" "cgl" "2003" "6"    ""  ""   NA
[4,] "1887" "cgl" "2003" "6"    "m" "ad" NA
```

Для успешной обработки данных необходимо следить, чтобы таблица содержала данные нужных типов, особенно в ключевых полях. Числовые поля должны иметь тип `num` или `int`, а текстовые – `Factor`. При обнаружении ошибок данные следует дополнить или исключить, тип данных – изменить. Можно, например, внести изменения в среде MS Excel и заново экспортировать рабочий файл.

В нашей таблице подозрительными оказываются два поля. Длина хвоста (`lc`) почему-то оказывается целой, хотя другие промеры имеют дробные значения. Проверка показала отсутствие ошибки, хвост измерялся в мм, тело – в см. Столбец для зрелости (`fer`) особей определен как фактор, хотя должен нести целочисленные значения. Оказалось, в базе имеются не только числовые значения, но и пробел. Выявить эти значения можно, например, функциями **unique()** и **table()**.

```
unique(mm$fer)
[1] ad  sad     juv
Levels:  ad juv sad
```

```
table(mm$fer)
     ad juv sad
  8  45   4   5
```

Оказалось, что для восьми особей зрелость не определена. Эти пробелы R воспринял как текст и сформировал массив типа `Factor`. Такие пробелы в данных нужно или ликвидировать, или заполнить. Технологии рассмотрены на стр. 74–80.

Классы объектов

После чтения базы данных класс нашего объекта (двумерного массива) был определен как `data.frame`. Такие объекты имеют важную особенность – все включенные в него векторы имеют одинаковое число полей (они образуют ряды). Именно с таким типом объектов чаще всего имеет дело биолог, их просто обрабатывают статистические функции. Основная причина состоит в том, что отдельные числовые значения должны иметь понятные идентификаторы (вид, пол и пр.), чтобы по этим признакам можно было формировать выборки, порознь вычислять статистическое параметры, сравнивать друг с другом и пр.

Из других классов объектов нужно отметить Список (`list`), который также составлен из совокупности векторов разных типов, но число значений в них может существенно отличаться. Еще один класс объектов – матрица (`matrix`) – важный, но специфический математический объект, с ними мы практически не будем иметь дела. Иногда удобно создавать объекты этого типа с помощью функции **`array()`**, а затем преобразовывать в класс `data.frame`, более удобный для манипуляций.

Изменение класса объекта

Для определения, к какому классу относится данный массив, используются функции вида **`is.…()`**.

```
is.list(mm)
[1] TRUE
is.data.frame(mm)
[1] TRUE
is.matrix(mm)
[1] FALSE
```

Когда результат расчетов выводится в виде *матрицы*, для продолжения расчетов ее приходится переводить в формат *таблицы*. Для изменения класса объекта служат функции вида **`as.…()`**.

```
array(c(1:24),c(4,6))
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    9   13   17   21
[2,]    2    6   10   14   18   22
[3,]    3    7   11   15   19   23
[4,]    4    8   12   16   20   24
```

```
is.matrix(array(c(1:24),c(4,6)))
[1] TRUE
```

Здесь с помощью функции **array()** сформирована матрица из чисел диапазона 1:24, содержащая 4 строки и 6 столбцов. Трансформация ее в объект класса таблица выполняется функцией **data.frame()**.

```
as.data.frame(array(c(1:24),c(4,6)))
  X1 X2 X3 X4 X5 X6
1  1  5  9 13 17 21
2  2  6 10 14 18 22
3  3  7 11 15 19 23
4  4  8 12 16 20 24
is.matrix(as.data.frame(array(c(1:24),c(4,6))))
[1] FALSE
is.data.frame(as.data.frame(array(c(1:24),c(4,6))))
[1] TRUE
```

Структура таких массивов отличается, как и способы обработки.

```
str(array(c(1:24),c(4,6)))
int [1:4, 1:6] 1 2 3 4 5 6 7 8 9 10 ...
```

```
str(data.frame(array(c(1:24),c(4,6))))
'data.frame':   4 obs. of  6 variables:
 $ X1: int  1 2 3 4
 $ X2: int  5 6 7 8
 $ X3: int  9 10 11 12
 $ X4: int  13 14 15 16
 $ X5: int  17 18 19 20
 $ X6: int  21 22 23 24
```

Изменение имен колонок и рядов

Когда при создании массивов имена полей или рядов оказываются невыразительными или длинными, их можно заменить на более удобные. Имена и рядов, и полей читает функция **dimnames()**, только рядов – **rownames()**, только колонок – **colnames()** и **names()**.

```
names(mm)
 [1] "n"      "spic"   "year"   "season" "sex"    "fer"    "age"
 [8] "yeseli" "line"   "w"      "lt"     "lc"     "lp"
colnames(mm)
 [1] "n"      "spic"   "year"   "season" "sex"    "fer"    "age"
 [8] "yeseli" "line"   "w"      "lt"     "lc"     "lp"
```

```
rownames(mm)
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12"
[13] "13" "14" "15" "16" "17" "18" "19" "20" "21" "22" "23" "24"
[25] "25" "26" "27" "28" "29" "30" "31" "32" "33" "34" "35" "36"
[37] "37" "38" "39" "40" "41" "42" "43" "44" "45" "46" "47" "48"
[49] "49" "50" "51" "52" "53" "54" "55" "56" "57" "58" "59" "60"
[61] "61" "62"
dimnames(mm)
[[1]]
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10" "11" "12"
[13] "13" "14" "15" "16" "17" "18" "19" "20" "21" "22" "23" "24"
[25] "25" "26" "27" "28" "29" "30" "31" "32" "33" "34" "35" "36"
[37] "37" "38" "39" "40" "41" "42" "43" "44" "45" "46" "47" "48"
[49] "49" "50" "51" "52" "53" "54" "55" "56" "57" "58" "59" "60"
[61] "61" "62"
[[2]]
 [1] "n"      "spic"   "year"   "season" "sex"    "fer"    "age"
 [8] "yeseli" "line"   "w"      "lt"     "lc"     "lp"
```

Для всех имен тип данных – `char` (текст). С помощью функций **`colnames()`** и **`rownames()`** можно заменить текущие имена колонок (здесь строчными буквами от a до m) и номера строк на новые.

```
colnames(mm)<-letters[1:13] # строчные буквы
rownames(mm)<-rev(dimnames(mm)[[1]]) # обратный порядок [[1]]
head(mm)
    a   b   c   d e   f g     h     i   j  k  l  m
62 976 cgl 2001 6 m  ad 9 200106002 2 24.0 95 40 16
61 977 sar 2001 6 f  ad 1 200106002 2 12.4 71 36 12
60 978 cgl 2001 6 m sad 2 200106002 2 24.4 91 39 16
```

Циклы и функции

Несмотря на ориентированность R на обработку векторов, программы могут быть достаточно большими с повторяющимися наборами команд, выполняющими стереотипные действия. Для структурирования и сжатия текста кода служат функция организации цикла **`for()`** и функция создания пользовательских функций **`function()`**.

Цикл – это повторное выполнение одних и тех же команд (функций). Число циклов задается счетчиком (по умолчанию – целым числом), которое при выполнении очередного цикла увеличивается на заданный шаг (по умолчанию – на единицу). Стартовое и конечное значения счетчика и его шаг – это и есть аргументы функции **`for()`**. Использование циклов в коде R можно считать «дурным тоном», поскольку для выполнения повторяющихся операций в R есть большое множество функций. Однако при первичном создании кода порой проще написать цикл, чем ломать голову над другим способом структурирования программы; апробировав работоспособность «дубо-

вого» кода, в окончательном варианте можно придать ему R-изящество. Подсказку для функции **for()** нужно искать, введя в консоли **?Control**.

Цикл перебора 7 случайных чисел выглядит так:

```
for(i in 7:1){print(round(runif(i),1))}
[1] 0.4 0.3 0.6 1.0 0.5 0.5 0.9
[1] 0.9 0.7 0.6 0.3 0.5 0.4
[1] 0.6 1.0 0.3 0.2 0.8
[1] 0.7 0.6 0.1 0.3
[1] 0.6 0.5 0.7
[1] 0.1 0.7
[1] 0.9
```

В качестве аргументов в круглых скобках указываются имя счетчика (i) и последовательность перебираемых значений счетчика (у нас – от 7 до 1). В фигурных скобках заключено тело цикла – множество циклически выполняемых команд. В примере – вывод на консоль все меньшего числа округленных случайных чисел (обратный отсчет).

Когда же может пригодиться функция цикла? В алгоритмических языках он использовался при обработке двумерных массивов. Решим задачу поиска средних значений массы тела зверьков всех видов, представленных в нашей базе данных. Для организации цикла нужно знать, сколько всего видов содержит база (ns), и предварительно организовать массив, в который будут записаны искомые средние по массе (mw). Кроме того, нужно избавиться от пробелов (NA) в данных с помощью функции **na.omit()**. Получаем следующий код:

```
ns<-length(sp<-unique(mm$spic)); mw<-c(1:ns)
for(i in 1:ns){mw[i]<-(mean(mm[mm$spic==sp[i],10],na.rm=TRUE))}
data.frame(sp=sp,M=mw)
   sp        M
1 cgl 23.23261
2 sar 11.00000
3 smi  6.70000
4 sbe 10.66667
5 mag 42.90000
```

Решаем ту же задачу без цикла. Как видно, это решение много проще.

```
data.frame(sp=sp,M=tapply(mm[,10],mm[,2],mean,na.rm=TRUE))
     sp        M
cgl cgl 23.23261
mag sar 42.90000
sar smi 11.00000
sbe sbe 10.66667
smi mag  6.70000
```

Рассмотрим задачу, которая часто возникает при обработке данных методами ресамплинга. Нужно создать диаграмму распределения средних арифметических, рассчитанных для 100 случайных выборок объемом по 10 вариант, извлеченных методом рандомизации с возвратом из небольшого (4 варианты) массива массы тела бурозубок.

При решении задачи с помощью цикла предварительно создаем массив для средних (`mss`). Задаем 100 циклов. В теле цикла из исходной выборки массы тела бурозубок без пропусков (**`na.omit(mm[mm$spic=='sar',10])`**) извлекаем по 10 значений, которые могут повторяться (**`sample(…,10, replace=TRUE)`**), для них рассчитываем среднюю (**`mean()`**) и заносим в массив `mss`. В заключение строим сглаженную кривую распределения.

```
ns<-100 ; mss<-1:ns
for(i in 1:ns)
{
mss[i]<mean(sample(na.omit(mm[mm$spic=='sar',10]),10,replace=TRUE))
}
plot(density(mss))
```

Решение задачи без использования цикла включает создание длинного ряда значений (**`sample(…,1000,...)`**), случайно извлеченных из исходной выборки, формирование из этого вектора двумерного массива с размерностью 100 рядов, 10 столбцов (**`array(...c(100,10)`**), расчет средних по строкам (**`apply(...,1,mean)`**) и построение диаграммы сглаженных частот (**`plot(density(...)`**).

```
plot(density(apply(array(
sample(na.omit(mm[mm$spic=='sar',10]),1000,replace=TRUE),
c(100,10)),1,mean)),main='')
```

В данном случае оба пути построения кода по своей сложности примерно равнозначны и нет очевидных преимуществ у какого-либо из них. Пожалуй, есть только один случай, когда применение цикла неизбежно – построение имитационных моделей (стр. 100–106).

Функция пользовательская – это совокупность команд на языке R любой сложности, выполнить которые можно, всего лишь прописав имя функции и необходимые аргументы. По способу вызова пользовательская функция ничем не отличается от базовых функций R, и многочисленные дополнительные пакеты языка R написаны именно как множества пользовательских функций на языке R. Подсказка для функции **`function()`** выводится по запросу в консоли **`?Control`**.

При построении пользовательских функций важно видеть два этапа работы. Во-первых, нужно создать эту функцию в отдельном блоке кода на R и выполнить записанное множество команд. При этом код данной функции будет загружен в память компьютера и готов к использованию, но никаких действий с данными выполнено не

будет. Все функции лучше всего записывать в самом начале программы, чтобы еще до начала манипуляции с данными они были готовы к работе.

Во-вторых, в нужных местах программы можно многократно вызывать функцию по имени (в том числе меняя значения аргументов); в этом случае команды функции сработают и будут *возвращать* предусмотренный результат расчетов.

Рассмотрим простой пример функции деления числа пополам. Здесь с помощью базовой функции **function()** строится пользовательская функция с именем **ff** и с заключенным в круглые скобки единственным аргументом **a**. Тело функции ограничено фигурными скобками {} и включает как некие действия над данными, так и ключевое слово **return()**, которое обозначает тот результат, который функция должна *возвращать* при ее вызове. В данном случае *результатом* будет частное от деления заданного числа на два **a<-a/2.**

Запись **ff(x)** означает *вызов* заданной функции, которая правильно выполнила деление числа 50 на 2 и возвратила число 25.

```
ff<-function(a) {a<-a/2; return(a)}
x<-50
ff(x)
[1] 25
```

При построении и использовании функций важно *различать внешние и внутренние переменные*. Функция – это программа в программе; код функции совершенно изолирован от кода основной программы, он имеет свою независимую область памяти. Имена переменных (величин, векторов...) внутри функции относятся к одним ячейкам памяти, а имена внешних переменных – к другим, поэтому даже при совпадении имен внутренних и внешних переменных они соответствуют разным областям памяти. Короче говоря, из внешней программы нельзя вызвать и посмотреть содержимое внутренней переменной данной функции. Это значит, что *внешним и внутренним переменным необходимо назначать разные имена*.

Наш пример пояснит это противоречие.

```
ff<-function(a){a<-a/2; return(a)}
a<-50
ff(a)
[1] 25
a
[1] 50
```

Казалось бы, в теле функции в ячейку **a** заносится половина исходного числа, т. е. в ней должно быть число 25. Но вызов из внешней программы показывает, что там по-прежнему число 50. Несмотря на то что внутренняя переменная уменьшается в два раза, внешняя переменная осталась неизменной! От этой путаницы совершенно необходимо избавляться, назначая внешним и внутренним переменным разные имена. Поль-

зовательские функции во многих случаях удобны, а в некоторых – необходимы. В частности, при использовании функций оптимизации (`nlm()`, `optim()` и др.) в список их аргументов необходимо включать заранее составленные пользовательские функции, вычисляющие *невязку*. Пример рассмотрен на стр. 100–106.

ДИАГРАММЫ

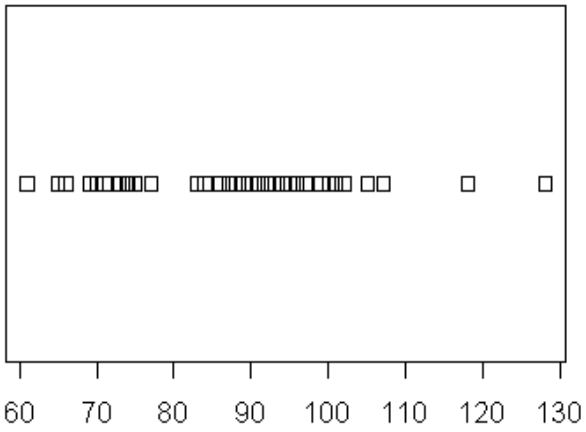
Построение диаграмм необходимо для первичного визуального анализа характера варьирования данных, для поиска ошибок, обнаружения зависимостей и наглядного представления результатов расчетов.

Базовые функции построения диаграмм

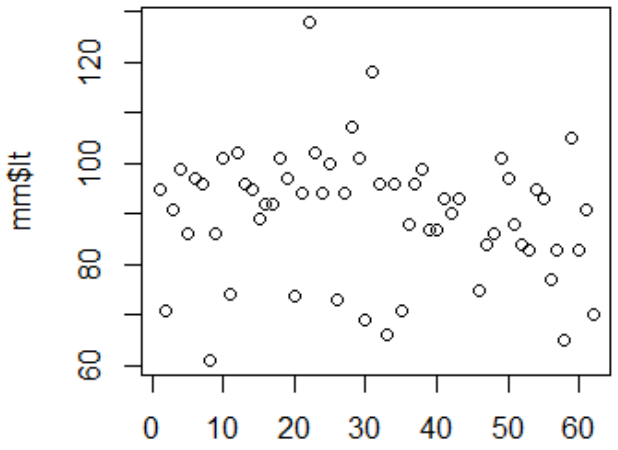
Данные числового вектора можно представить в виде диаграммы разброса (`stripchart()`), графика (`plot()`), «ящика с усами» (`boxplot()`) и частотной гистограммы (`hist()`). Для вывода готового рисунка достаточно в скобках указать только имя объекта, остальным аргументам программа задает значения по умолчанию; они приведены в справке (`?...`) и могут быть изменены.

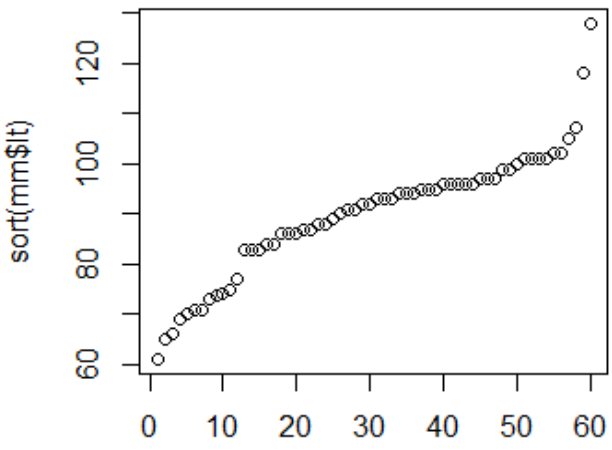
Например, на диаграммах для длины тела полевок (`mm$lt`) сразу выявились особенности выборки. Все значения распались на две большие группы, но выделилось и два экстремальных значения – минимальное и максимальное.

```
stripchart(mm$lt)
```

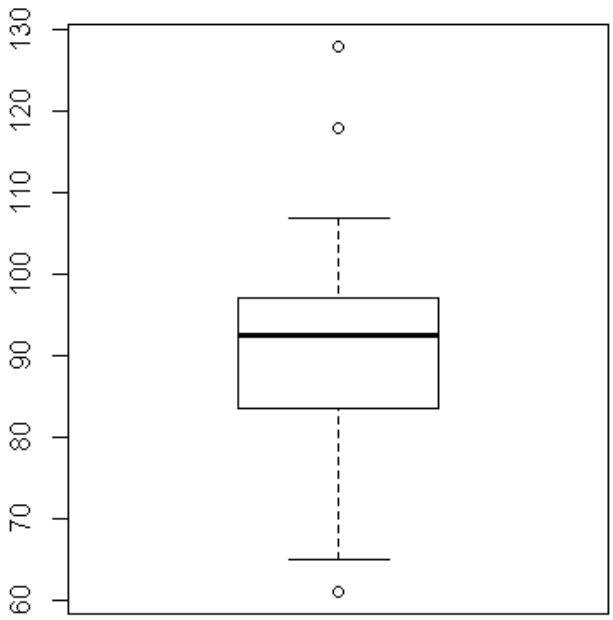



```
plot(mm$lt)
plot(sort(mm$lt))
```

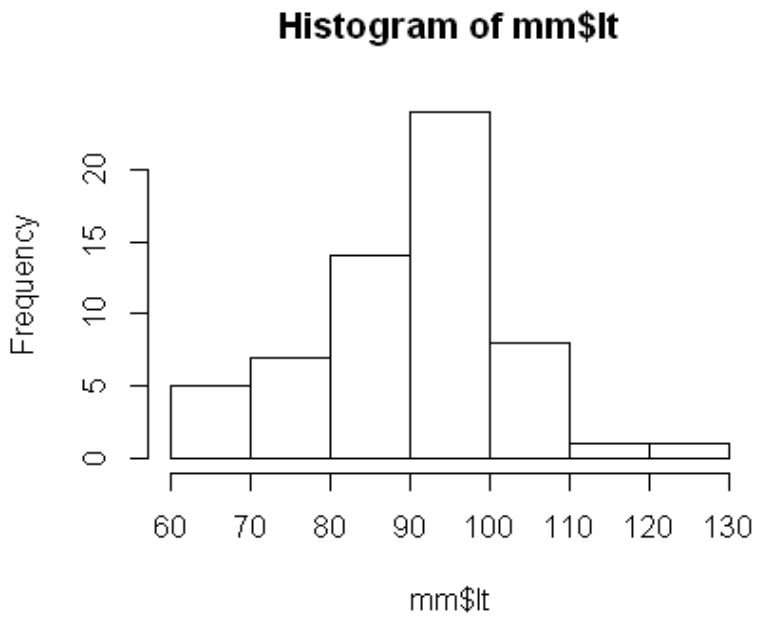




```
boxplot(mm$lt)
```




```
hist(mm$lt)
```



После подсчета частоты встречаемости разных видов (**`table()`**) эта особенность стала понятной – минимальное значение относится к малой бурозубке (`smi`), максимальное – к темной полевке (`mag`), а две группы – это взрослые рыжие полевки (`cgl`), а также молодые полевки, мышовка (`sbe`) и обыкновенная бурозубка (`sar`).

```
table(mm$spic)
cgl mag sar sbe smi
53   1   4   3   1
```

Построенные диаграммы вполне годятся для рабочего анализа выборок, но для публикаций они должны быть соответствующим образом перестроены путем изменения значений аргументов. Новые аргументы функции записываются в скобках через запятую после имени объекта.

Универсальные аргументы

Подписи осей абсцисс и ординат задаются аргументами **`xlab`** и **`ylab`** (например, **`xlab="Число особей"`**).

Диапазон значений на осях выставляют с помощью аргументов **`xlim`** и **`ylim`**, указывая векторы с двумя крайними значениями. Например, запись **`ylim=c(50, 120)`** расширяет диапазон для оси ординат.

Цвет элементов диаграммы (**`col`**) можно задать его именем (серый – **`col="gray"`**), которое можно уточнить, вызвав список цветов, используемых в R командой **`colors()`**, или указать целочисленный HTML-код цвета (серый – **`col=8`**).

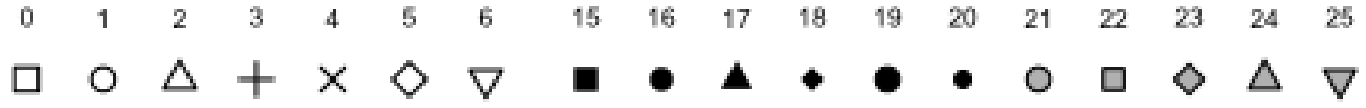
Размер элементов на поле диаграммы (точки, символы) регулирует аргумент **`cex`** (по умолчанию **`cex=1`**).

Величину подписей осей и значений на осях изменяют аргументы **`cex.lab`** и **`cex.axis`**.

Ориентацию чисел на осях меняет аргумент **`las:`** на осях ординат – **`las=1`**, абсцисс **`-`** (**`las=3`**) или обеих осях – (**`las=2`**).

Аргументы функции **`plot()`**

Форма маркера в поле диаграмм графика и разброса настраивается аргументом **`pch`** – (например, **`pch=4`**). Обычно используют 25 стандартных форм маркера (см. справку **`?pch`**). Символы 21–25 можно представить двумя цветами – контуры регулируются аргументом **`col`**, заливка – **`bg`** с теми же кодами цветов.



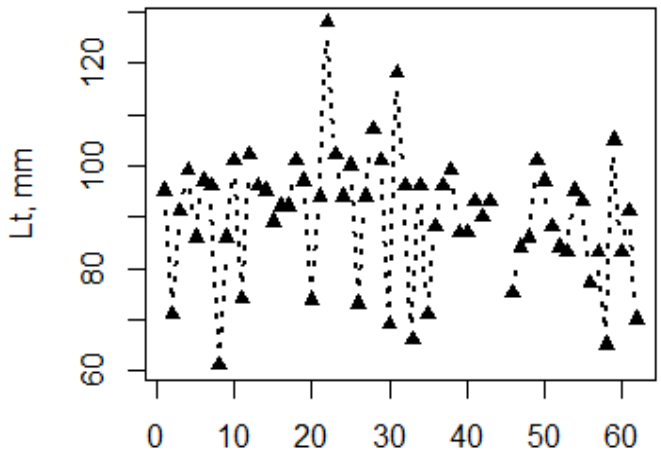
Способ налегания и выделения точек задают аргументом **`method`** (по умолчанию **`method="overplot"`** – нижние точки затираются, другие варианты – **`"stack"`** и **`"jitter"`**).

Тип графического отображения данных регулируется аргументом **`type`**. (по умолчанию **`type='p'`** – задана точка; **`'l'`** – линия, **`'b'`** – и линия, и точки, **`'n'`** – отсутствие отображения).

Тип линии задается аргументом **lty**, стандартная кодировка которого предусматривает 7 позиций (по умолчанию **lty=1** – сплошная линия, **0** – нет линии, **2** – пунктир, **3** – точечная; другие варианты рассмотрены в справке **?par**).

Толщину линий указывают с помощью **lwd** (по умолчанию **lwd=1**).

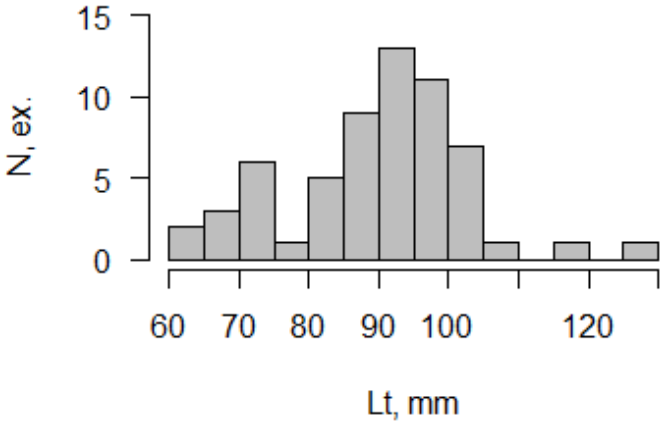
```
plot(mm$lt,type='b',ylab="Lt,mm",xlab='',lty=3,lwd=2,pch=17)
```



Аргументы функции **hist()**

Число столбцов гистограммы по умолчанию не задано, функция его рассчитывает, исходя из объема выборки, но его можно задать по своему усмотрению аргументом **breaks**.

```
hist (mm$lt, xlab="Lt, mm", ylab="N, ex.", ylim=c(0, 15), breaks=10, col="gray", las=1)
```

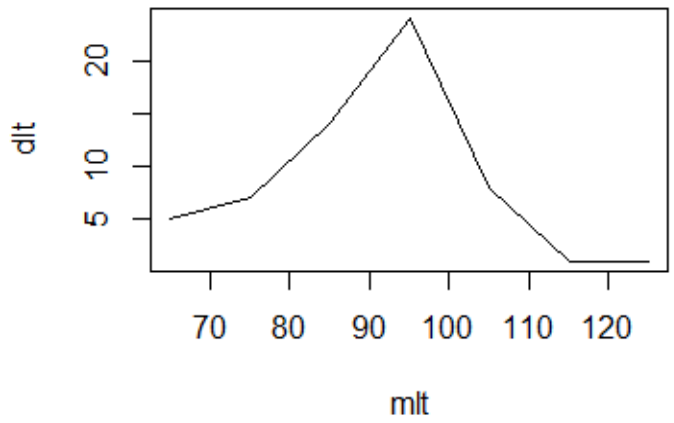


Функция **hist()** работает в два этапа. Сначала она рассчитывает характеристики столбчатой диаграммы, затем строит эту диаграмму в окне Grapgics. Вместо диаграммы можно вывести результаты обработки выборки, если изменить значение логического аргумента **plot=FALSE**.

```
hist(mm$lt,plot=FALSE)
$breaks
[1]  60  70  80  90 100 110 120 130
$counts
[1]  5  7 14 24  8  1  1
$density
[1] 0.008333333 0.011666667 0.023333333 0.040000000
[5] 0.013333333 0.001666667 0.001666667
$mids
[1]  65  75  85  95 105 115 125
$xname
[1] "mm$lt"
$equidist
[1] TRUE
attr(,"class")
 [1] "histogram"
```

Используя эти данные, гистограмму можно отобразить как полигон частот с помощью функции **plot()**.

```
mlt<-hist(mm$lt)$mids
dlt<-hist(mm$lt)$counts
plot(mlt,clt,type='l')
```

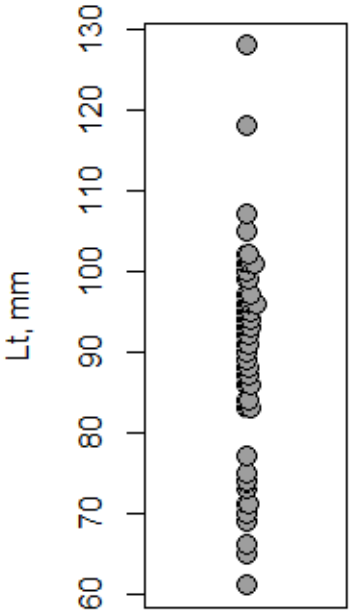


Единицы на оси ординат по умолчанию задаются как частоты – число объектов, попавших в разные классы гистограммы, т. е. для построения гистограммы используются данные вектора **$counts**. Чтобы построить диаграмму с относительными частотами, т. е. использовать вектор **$density**, следует изменить аргумент **freq=FALSE**.

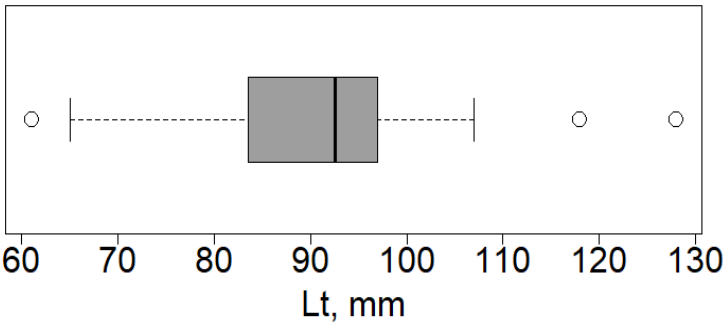
Аргументы функций `boxplot()` *и* `stripchart()`

Ориентацию рисунка изменяют с помощью логических аргументов (по умолчанию для функции `stripchart()` принято **`horizontal=TRUE`**, для `boxplot()` – **`vertical=TRUE`**). Длину «усов» и «ящика» регулируют аргументом **`range`** (по умолчанию **`range=1.5`**, при условии **`range=0`** «усами» указывается весь диапазон варьирования признака). Вместо диаграммы можно вывести результаты обработки выборки, если изменить значение логического аргумента **`plot=FALSE`**.

```
stripchart (mm$lt, ylab="Lt, mm",
method="stack", cex=1.5, vertical=TRUE,
pch=21, bg=8)
```




```
boxplot(mm$lt, xlab="Lt, mm", cex=2, cex.axis=2,
cex.lab=2, cex.main=2, col=128, horizontal=TRUE)
```

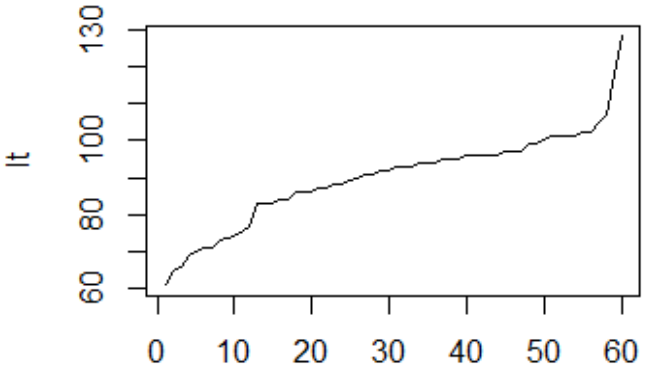


Функции наложения диаграмм

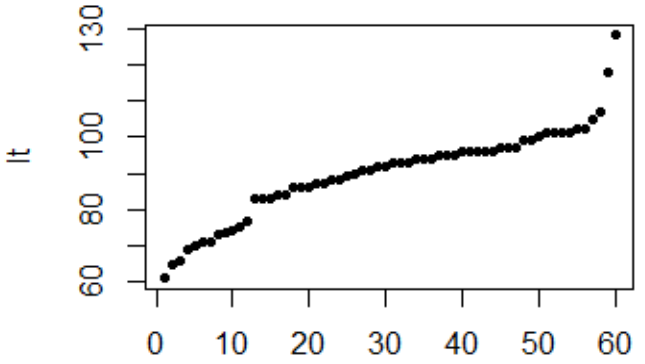
На диаграммы, построенные с помощью функций **`plot()`**, **`hist()`**, **`boxplot()`**, можно добавить другую графическую или текстовую информацию с помощью функций **`lines()`**, **`points()`**, **`text()`**.

Функция **`plot()`** всегда заново создает поле диаграммы, на которое можно не выводить график, если в аргументах указать **`type='n'`**. После этого с помощью дополнительных функций можно нанести линии (**`lines()`**), точки (**`points()`**) или текстовую информацию (**`text()`**), например, сами значения анализируемого признака. В примере изучаемые значения длины тела предварительно были лишены пробелов (**`na.omit()`**), отсортированы в порядке возрастания (**`sort()`**), округлены до одного десятичного знака после (**`round()`**).

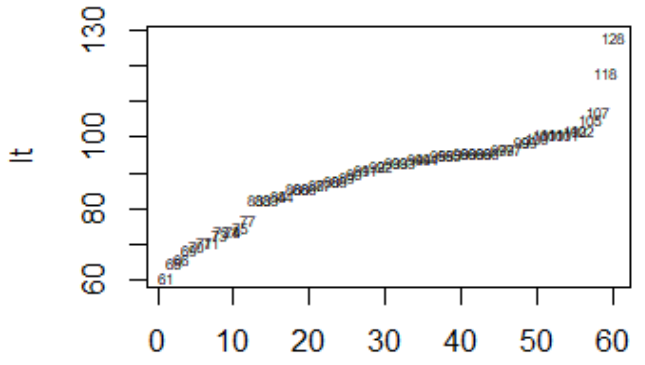
```
lt<-round(sort(na.omit(mm$lt)),1)

plot(lt,type='n')
lines(lt)
```




```
plot(lt,type='n')
points(lt,pch=20,cex=1)
```

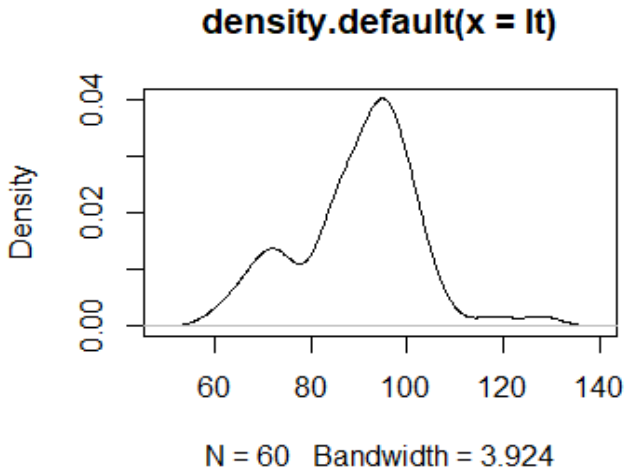



```
plot(lt,type='n')
text(lt,labels=lt,cex=0.5)
```



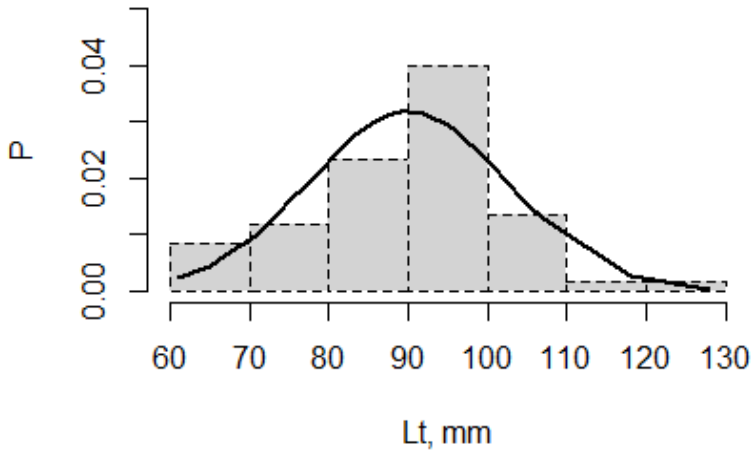
Вместо гистограммы или полигона частот иногда имеет смысл построить сглаженную кривую плотности вероятности; ее значения рассчитывает функция **`density()`**, а саму линию строит функция **`plot()`**.

```
lt<-round(sort(na.omit(mm$lt)),1)
plot(density(lt))
```



Для визуального сопоставления реальных замеров с теорией имеет смысл наложить теоретический график на частотную гистограмму, например график кривой нормального распределения. Теоретические частоты рассчитываются функцией **dnorm()** с использованием параметров выборки – средней арифметической (**mean(lt)**) и стандартного отклонения (**sd(lt)**). Функция рассчитывает относительные частоты нормального распределения, по этой причине гистограмма также должна так же отображать доли, поэтому добавляем аргумент **freq=FALSE**. Построив гистограмму (**hist()**), накладываем на нее нормальную кривую с помощью дополнительной функции **lines()**.

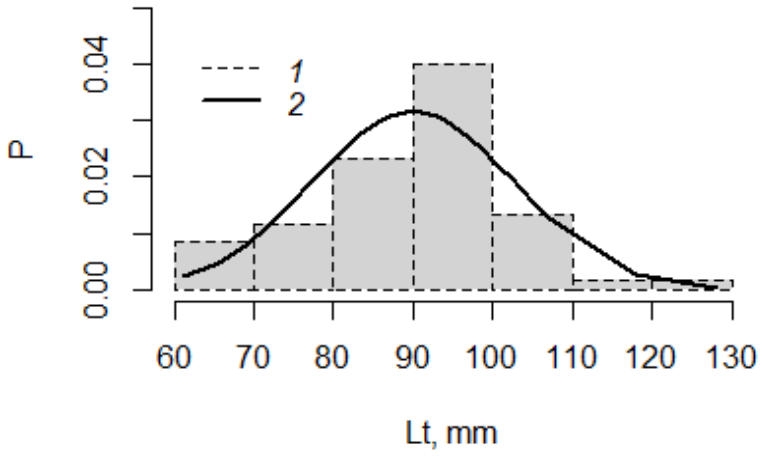
```
hist(lt, main=NULL, xlab="Lt, mm", ylab="P", ylim=c(0, 0.05), lty=2, lwd=1,
breaks=5, freq = FALSE)
lines(lt, dnorm (lt, mean(lt), sd(lt)), lty=1, lwd=2)
```



Оформление легенды

Поскольку линий стало две, необходимо оформить легенду – цифровое обозначение разных графических элементов с помощью функции **`legend()`**. Легенда строится отдельно от диаграммы и автоматически не собирает информацию о ее элементах, все аргументы программируются отдельно.

```
legend(60, 0.045, lty=c(2,1), lwd=c(1,2), box.lty=0,
legend=c(expression(italic('1')), expression(italic('2'))))
```



В примере местоположение блока подписей задается аргументами координат левой (**`x`**) верхней (**`y`**) точки рамки легенды в единицах осей (**`x=60, y=0.045`**); или же можно указать область ее размещения с помощью ключевых слов **`"topright"`**, **`"topless"`**, **`"bottomleft"`** и др.

Графические элементы рисуем с помощью аргументов **`lty`** (тип линии) и **`lwd`** (толщина линии). Этим аргументам нужно присвоить вектор, содержащий значения, соответствующие характеристикам на диаграмме. В примере гистограмма отображена тонким пунктиром (**`lty=2, lwd=1`**), кривая – сплошной жирной линией (**`lty=1, lwd=2`**).

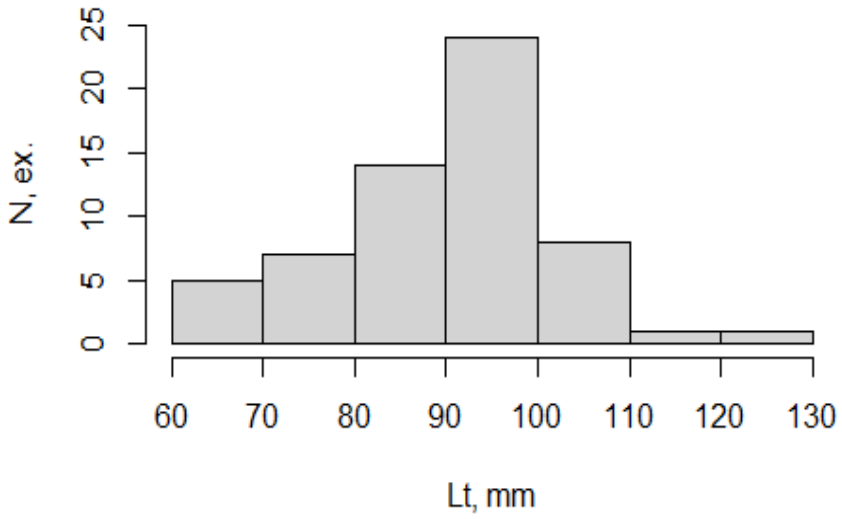
Рамка вокруг легенды по умолчанию рисуется сплошной линией; аргумент **`box.lty=0`** удаляет рамку.

В аргументе **`legend`** указаны подписи элементов диаграммы в форме ряда *текстовых* значений. Текст обязательно должен быть ограничен кавычками или апострофами. Здесь это **`'1'`** и **`'2'`** – по правилам многих изданий элементы легенды обозначают цифрами, которые расшифровывают в подрисуночной подписи. Для подписи курсивом нужно воспользоваться функцией **`expression()`**. Она вызывает выражения из списка символов математической графики (справка – **`?plotmath`** из пакета **`grDevices`**). Выражение для курсива (**`italic()`**) должно содержать в скобках выводимый текст.

Функции настройки осей

При формировании осей программа самостоятельно подбирает размах и шаг разметки.

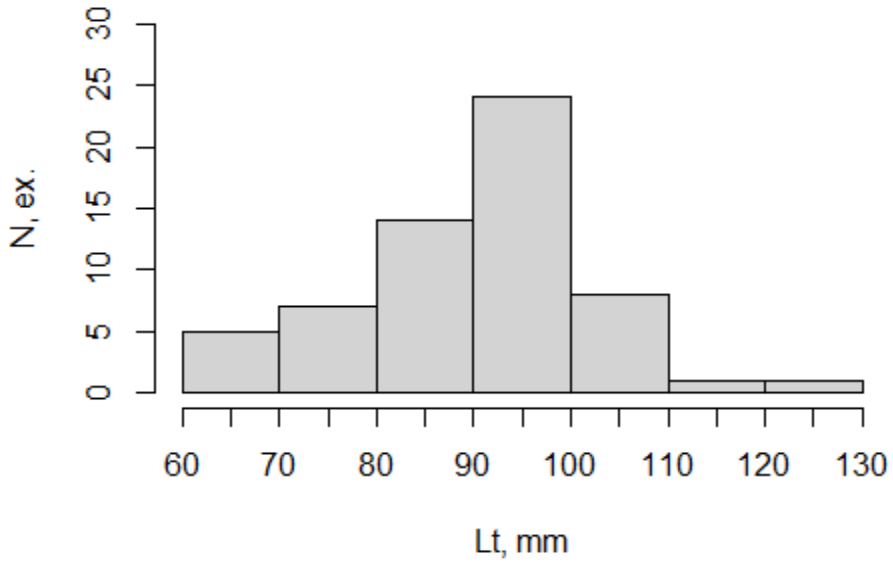
```
hist (mm$lt, main=NULL, xlab="Lt, mm", ylab="N, ex.")
```



Для управления осями служит функция **axis()**. Сначала в функции **hist()** следует отключить прорисовку осей – либо обеих (**axes=FALSE**), либо одно из них (**xaxt="n"** или **yaxt="n"**). Затем задать нужные аргументы функции **axis()**. Аргумент **side** определяет сторону графика, где будет помещена ось (**1** – снизу, **2** – слева, **3** – сверху, **4** – справа). Аргумент **at** указывает на вектор, в котором нужно записать диапазон и шаг разметки, что удобно сделать с помощью функции **seq()**. Аргумент определяет, где следует рисовать риски на оси и какие риски следует подписывать. Если риски должны стоять чаще, чем подписи, следует написать две функции, но у одной отключить нанесение подписей (**label=FALSE**).

```
hist (mm$lt, main=NULL, xlab="Lt, mm", ylab="N, ex.", ylim=c(0, 30), ax-
es=FALSE)
axis(side=1, at=seq(60,130,5),label=FALSE)
axis(side=1, at=seq(60,130,10))
axis(side=2, at=seq(0,30,5))
```

В примере для оси абсцисс риски заданы с интервалом 5 (**at=seq(60,130,5)**), а подписи заданы с интервалом 10 (**at=seq(60,130,10)**). Ось ординат в исходном рисунке имеет диапазон от 0 до 25. Чтобы увеличить его до 30 с помощью функции **axis()**, нужно ввести соответствующее расширение в функцию **hist()** (**ylim=c(0, 30)**), иначе на диаграмме ось будет обрезана.



Отображение нескольких векторов

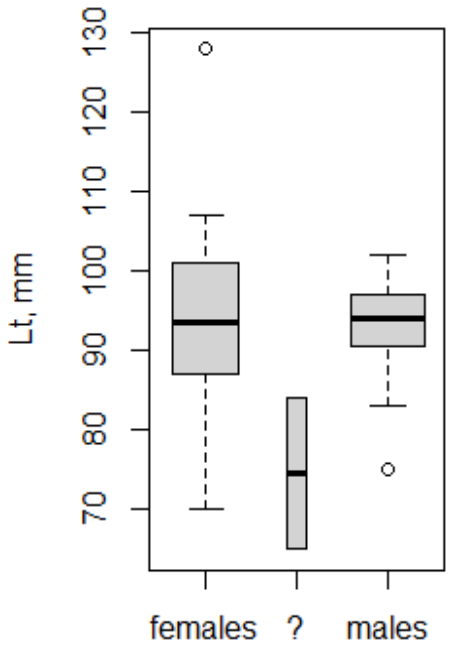
Диаграмма размаха – «ящик с усами»

В диаграмме размаха по умолчанию границы «ящика» выражают квартили 25 % и 75 %, и «усы» – еще по 50 % от квартилей, т. е. 12.5 % и 87.5% диапазона выборки. Анализ диаграммы может предварять решение о необходимости статистических сравнений, например, самцов с самками рыжей полевки (**`cgl`**). Для быстрого построения диаграммы следует воспользоваться функцией **`boxplot()`**.

```
head(cgl<-mm[mm$spic=='cgl',],3) [,c(2,5,11)]
  spic sex lt
1  cgl   m 95
3  cgl   m 91
4  cgl   m 99
boxplot(cgl$lt~cgl$sex, ylab="Lt, mm",xlab=NULL, varwidth=TRUE, at=c(2,1,3),
xaxt="n")
axis(side=1, at=seq(1,3), labels=c("females", "?", "males"))
```

Первый аргумент (**`formula`**) описывает структуру диаграммы – на первом месте указываются имя столбца, значения которого предстоит обрабатывать (**`cgl$lt`**) (признак), после знака **`~`** (тильда) на втором месте указывается имя столбца, содержащего ключевые значения для разделения данных на выборки (**`cgl$sex`**) (фактор). В базе данных представлены три группы (`$sex: Factor w/ 3 levels`): особи с неустановленным полом, самки и самцы. Логический аргумент **`varwidth=TRUE`** делает ширину прямоугольников пропорциональной квадратному корню объема выборок, что позволяет увидеть несущественный объем неопределенных особей. Числовой аргумент **`at`** использован для изменения последовательности расположения категорий вариант на рисунке, принятых по умолчанию: сначала самки (**`2`**), затем особи с неустановленным полом (**`1`**) и, наконец, самцы (**`3`**). Подписывание оси абсцисс блокировано аргументом

xaxt="n". Для этих целей использована функция **axis()** – сформирована шкала с тремя делениями и тремя подписями.



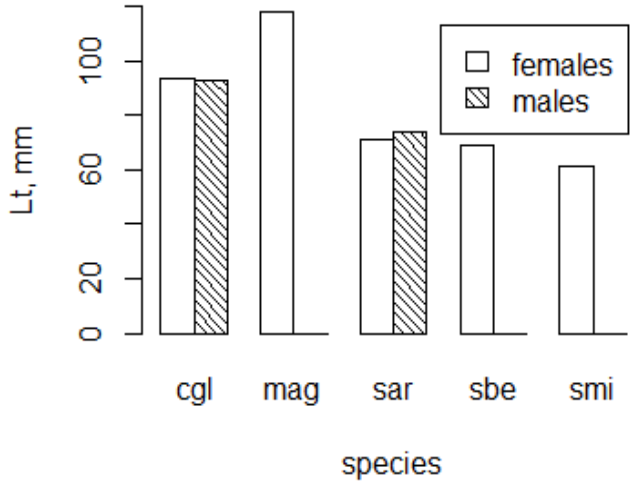
Столбчатые диаграммы

Столбчатые диаграммы выводят функцией **barplot()**. Например, нужно визуализировать отличия средних арифметических длины тела у разнополых особей пяти видов. Для этого создаем таблицу, состоящую их трех полей (**mm$spic**, **mm$sex**, **mm$lt**), и рассчитаем средние по заданным категориям. Перед расчетами исключим ячейки без чисел (**na.omit()**), округлим средние и оставим только вторую и третью строки, в которых есть данные по самкам и самцам.

```
mms<-na.omit(data.frame(mm$spic, mm$sex, mm$lt))
ltfm<-round(tapply(mms$mm.lt, list(mms$mm.sex, mms$mm.spic), mean), 1)[2:3,]
ltfm
    cgl    mag    sar    sbe    smi
f   93.5   118   71.0    68.7    61
m   93.1    NA   73.9     NA     NA
```

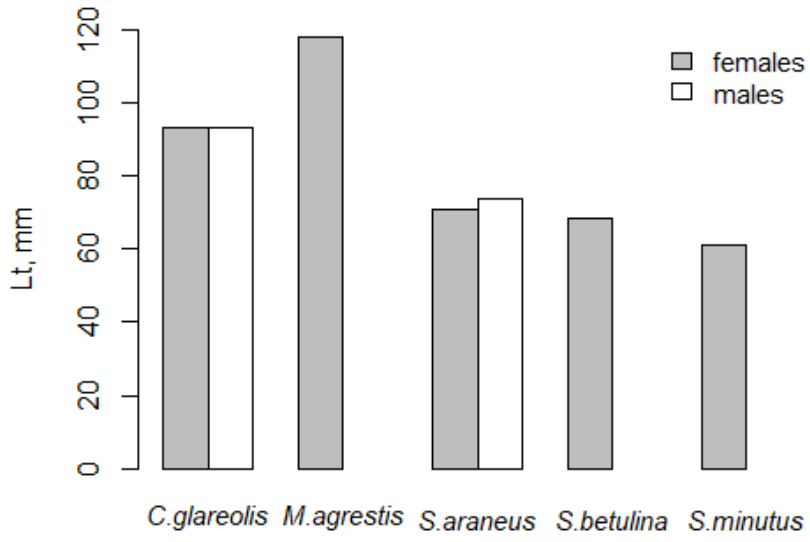
В функции **barplot()** аргумент **beside=TRUE** включает опцию распределения чисел по столбцам (по умолчанию выводятся диаграмма с накоплением). Для графического заполнения столбиков используются аргументы **density**, **angle** и **col**, которые задают плотность, угол наклона и цвет линий штриховки. Легенда оформлена с помощью аргумента **legend.text**.

```
barplot(ltfm, beside=TRUE, ylab="Lt, mm", xlab="species", ylim=c(0,120), density=c(0,25), angle=c(0,-45), col=1, legend.text =c("females","males"))
```



В дополнение к функции **barplot()** параметры вывода можно уточнить с помощью функции **legend()**. В примере аргументами **col**, **fill**, **names.arg** и **cex.names** заданы цвет фона столбцов и условных обозначений, названия столбцов и масштаб текста подписей.

```
barplot(ltfm, beside=TRUE, ylab="Lt, mm", ylim=c(0,120), col=c('grey','white'), names.arg=c(expression(italic("C.glareolis")), expression(italic("M.agrestis")), expression(italic("S.araneus")), expression(italic("S.betulina")), expression(italic("S.minutus"))), cex.names=0.9)
legend("topright", fill=c('pink','light blue'), legend=c("females", "males"), box.lty=0, cex=0.9)
```



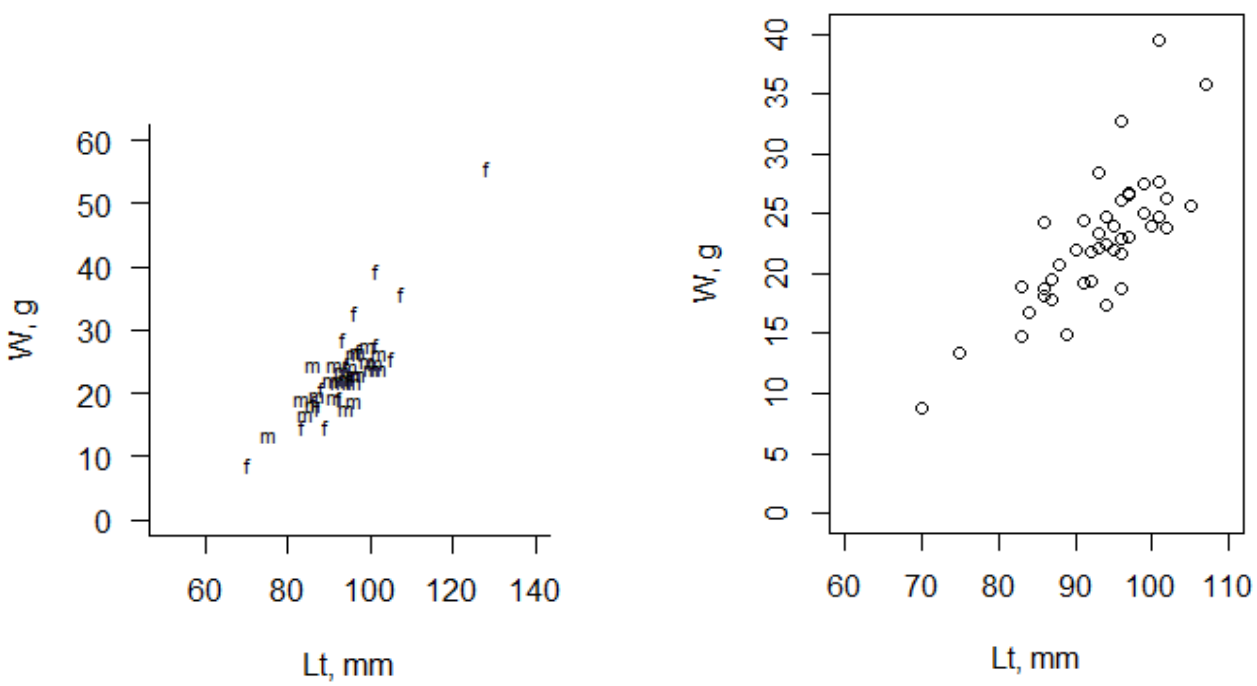
Диаграммы зависимостей

Диаграмму двумерного рассеяния строят с помощью функции **`plot()`**. Основные аргументы описаны выше. Вместо точек (**`type="n"`**) с помощью дополнительной функции **`text()`** на диаграмму можно наложить компактные (**`cex=0.7`**) маркеры половой принадлежности особей (**`labels=mm$sex`**).

```
plot(cgl$lt, cgl$w, xlab="Lt, mm", ylab="W, g", ylim=c(0, 60), xlim=c(50, 140),
bty="l", type="n", las=1)
text(cgl$lt, cgl$w, labels=cgl$sex, cex=0.7)
```

Для управления осями служит функция **`axis()`**. Используют те же аргументы, что и для функции **`hist()`** (стр. 51–52). Например, если нужно уменьшить и разметить ось ординат через пять единиц, ее сначала отключают, добавив **`yaxt='n'`** в список аргументов функции **`plot()`**, и создают новую ось (**`axis()`**) с делениями через пять единиц. Обратите внимание, что диапазоны значений в **`ylim`** и **`at`** должны быть одинаковыми.

```
plot(cgl$lt, cgl$w, xlab="Lt,mm", ylab="W,g", ylim=c(0,40), xlim=c(60,110),
yaxt='n')
axis(side=2, at=seq(0,40,5))
```



Используя другие дополнительные функции (**`lines()`**, **`points()`**), на рисунок можно добавить новые элементы – линии регрессии, доверительные интервалы, интервалы прогноза и т. д. Для этого создаем массив из двух колонок (**`cgl[...,c(11,10)]`**) из чисел без пропусков (**`na.omit()`**), предварительно упорядочив

ее по возрастанию длины тела (**order(cgl$lt)**) (для упрощения процесса рисования дополнительных линий).

```
head(cglr<-na.omit(cgl[order(cgl$lt),c(11,10)]),5)
   lt    w
62 70  8.7
46 75 13.4
53 83 18.9
60 83 14.7
47 84 16.7
```

Далее рассчитываем линейную регрессию **lm(cglr$w~cglr$lt)** зависимости массы от длины тела полевок и с помощью функции **predict()** вычисляем и заносим в массив **mod** модельные значения (**fit**) и границы доверительных (**"confidence"**) интервалов (**lwr** – нижняя и **upr** – верхняя границы).

```
head(mod<-predict(lm(cglr$w~cglr$lt),   interval="confidence"),5)
```

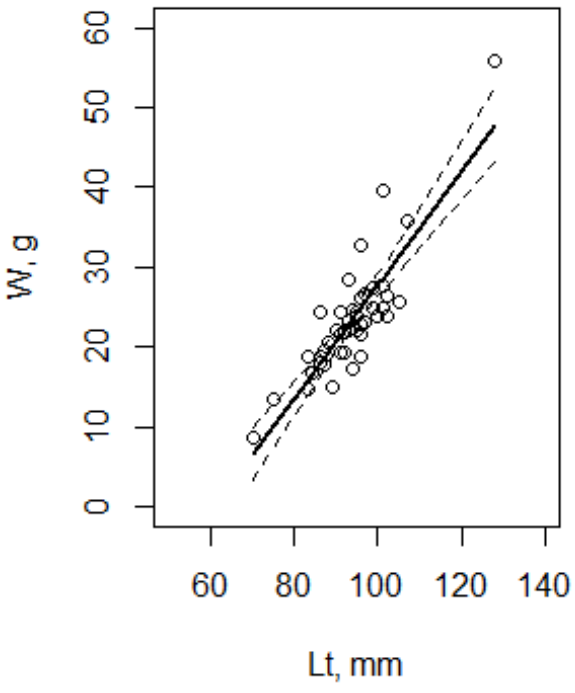
```
        fit       lwr       upr
1  6.592106  3.255182  9.929031
2 10.134834  7.402284 12.867385
3 15.803200 13.962248 17.644152
4 15.803200 13.962248 17.644152
5 16.511746 14.769788 18.253703
```

Для дальнейшей работы необходимо изменить класс массива **mod** – из матрицы сделать таблицей.

```
mod<-as.data.frame(mod)
```

Теперь остается создать точечную диаграмму (**plot()**) и с помощью функции **points()** наложить на нее графики трех рассчитанных числовых векторов.

```
plot(cgl$lt, cgl$w, xlab="Lt, mm",
ylab="W, g", ylim=c(0,60), xlim=c(50,
140))
points(cglr$lt,mod$fit,type='l',lwd=2)
points(cglr$lt,mod$lwr,type='l',lty=2)
points(cglr$lt,mod$upr,type='l',lty=2)
```



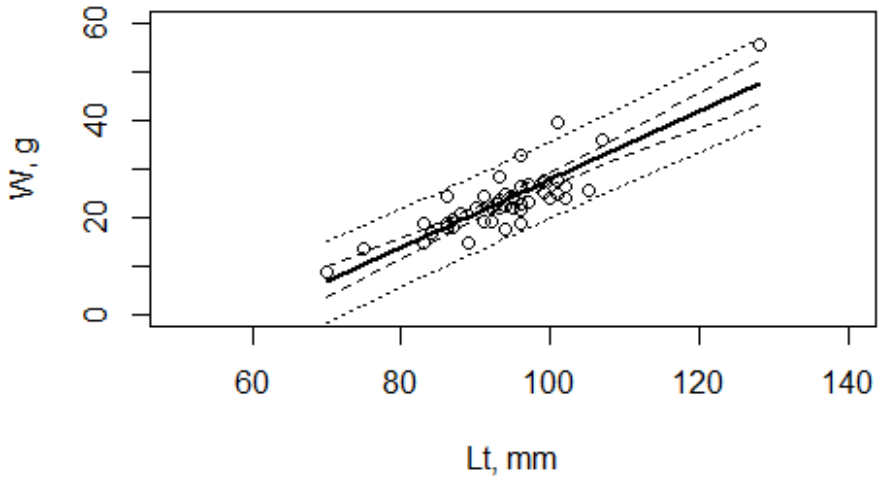
С задачей наложения нескольких графиков изящнее справляется функция **matplot()**, если дать разрешение в аргументе **add=TRUE**. Также удобно, что функция может работать с матрицей, которую создает функция **cbind()**. В примере кроме зоны доверительных (**"confidence"**) интервалов добавлен интервал прогноза (**"prediction"**).

```
mod<-cbind(predict(lm(cglr$w~cglr$lt), interval="confidence"),
predict(lm(cglr$w~cglr$lt), interval="prediction"))
head(mod,5)
```

```
        fit       lwr       upr       fit       lwr      upr
1  6.592106  3.255182  9.929031  6.592106 -1.888834 15.07305
2 10.134834  7.402284 12.867385 10.134834  1.872988 18.39668
3 15.803200 13.962248 17.644152 15.803200  7.791934 23.81447
4 15.803200 13.962248 17.644152 15.803200  7.791934 23.81447
5 16.511746 14.769788 18.253703 16.511746  8.522647 24.50084
```

```
plot(cgl$lt, cgl$w, xlab="Lt, mm", ylab="W, g", ylim=c(0, 60), xlim=c(50, 140))
matplot(cglr$lt,mod,type='l', lty=c(1,2,2,1,3,3), lwd=c(2,1,1,2,1,1),
col=c(1,1,1,1,1,1), add= TRUE)
```

Аргументы для типа (**lty**), толщины (**lwd**) и цвета (**col**) линий заданы по порядку для каждого из шести полей полученного массива **mod**.



ВЫБОРКИ ИЗ БАЗ ДАННЫХ

Типичная выборка для экологического анализа – это таблица, состоящая из двух полей, двух векторов – зависимой переменной (y) и независимой переменной (x), записи для которых были предварительно отобраны как относящихся к определенной группе объектов (по виду, полу, статусу и пр.). На основании такой выборки можно оценить статистическую зависимость между переменными. Получается, что в процессе формирования выборки большая база данных должна быть уменьшена как по числу строк, так и по числу колонок.

Кроме этого, часто возникает задача рассчитать некие производные значения для каждой записи, например отношения, нормированные отклонения, индексы и пр. Это новые переменные могут войти в выборку или послужить ключом для отбора записей. В частности, отношение длины хвоста к длине тела у змей служит хорошим критерием для проверки правильности определения пола. Одним из вариантов таких вторичных значений являются выборки особенных значений – минимальных, максимальных, точек перегиба и положения скачков изучаемых признаков.

Наконец, в состав выборки из базы данных обычно включают данные из разных таблиц. Так, если таблица для особей мелких млекопитающих показывает, сколько зверьков было отловлено в разные линии давилок, то таблица для регистрации линий показывает объем промысловых усилий. Только объединение этих данных позволяет рассчитать оценки относительной численности животных в тот или иной период, т. е. построить выборку с полями «год» и «численность».

Пятый уровень иерархии процесса био-экологического исследования включает пять процедур: проверка данных, отбор полей, объединение таблиц, построение фильтра, отбор записей по ключам, окончательный отбор полей, формирование выборки.

Отбор полей из таблицы при чтении файла

Таблицы базы данных могут включать несколько десятков полей, из которых большинство не нужны для формирования выборки. Нужные поля можно отобрать для работы после чтения файла. Рабочая таблица займет в памяти меньше места и будет быстрее обрабатываться, кроме того, с малым числом переменных проще работать. Для этого нужно определить позиции столбцов. В нашем примере файл **mamm.csv** содержит данные отлова зверьков. Читаем файл, выводим в консоль, выбираем нужные поля, записываем в рабочий массив. Чтобы не выводить в консоль весь массив, удобно использовать функции `head()` и `tail()`, которые визуализируют только заданное число верхних и нижних строк массива (по умолчанию – 6, в примере – 3).

```
head(mm<-read.csv("mamm.csv"),3)
```

```
    n spic year season sex fer age    yeseli line  w  lt lc lp
1 976  cgl 2001      6   m  ad   9 200106002   2 24.0 95 40 16
2 977  sar 2001      6   f  ad   1 200106002   2 12.4 71 36 12
3 978  cgl 2001      6   m sad   2 200106002   2 24.4 91 39 16
```

Цель анализа состоит в подсчете зверьков, попавших в разные линии, значит, нам не нужны многие индивидуальные характеристики особей, кроме вида. Из исходной таблицы оставляем только номер особи, вид, год, месяц, пол, составной индекс, и номер линии, в которую попался зверек (поля имеют индексы 2, 3, 4, 5, 8, 9 – порядковые номера соответствующих полей).

```
smm<-mm[,c(2,3,4,5,8,9)]
head(smm,3)
```

```
  spic year season sex    yeseli line
1  cgl 2001      6   m 200106002    2
2  sar 2001      6   f 200106002    2
3  cgl 2001      6   m 200106002    2
```

Эту процедуру можно упростить. Уже на стадии чтения данных из файла можно определиться, какие поля оставить в качестве ключа, какие в качестве данных для расчетов. Сначала с помощью функции **str()** рассматриваем структуру таблицы и решаем, какие столбцы нужны.

```
str(read.csv("mamm.csv"))
```

```
'data.frame':   62 obs. of  13 variables:
$ n     : int  976 977 978 979 980 981 982 983 984 985 ...
$ spic  : Factor w/5 levels "cgl","mag","sar",...: 1 3 1 1 1 1..
$ year  : int  2001 2001 2001 2001 2001 2001 2001 ...
$ season: int  6 6 6 6 6 6 6...
$ sex   : Factor w/ 3 levels "","f","m": 3 2 3 3 3 3 2...
$ fer   : Factor w/ 4 levels "","ad","juv",..: 2 2 4 4 2 2 2...
$ age   : int  9 1 2 2 9 9 9 1 2 2 ...
$ yeseli: int  200106002 200106002 200106002 200106002  ...
$ line  : int  2 2 2 2 2 2 1 1 2 2 ...
$ w     : num  24 12.4 24.4 25 24.3 23 32.7 6.7 18.7 27.6 ...
$ lt    : num  95 71 91 99 86 97 96 61 86 101 ...
$ lc    : int  40 36 39 43 45 44 41 32 40 44 ...
$ lp    : num  16 12 16 17 16 17 16 9 16 16 ...
```

Оставляем те же поля, читаем данные в массив **mm** и сразу рассматриваем результат.

```
head(smm<-read.csv("mamm.csv")[,c(2,3,4,5,8,9)],4)
```

```
  spic year season  sex  yeseli    line
1  cgl 2001      6   m 200106002    2
2  sar 2001      6   f 200106002    2
3  cgl 2001      6   m 200106002    2
4  cgl 2001      6   m 200106002    2
```

Отбор записей из одной таблицы по ключам

Выборка из базы данных формируется в процессе сравнения каждой записи с определенным критерием. Отбираются только те записи, для которых выполняются условия отбора, остальные не включаются в результат.

Дано: массивы базы данных (большие таблицы)

определение ключей
определение критериев
построение фильтра
отбор (фильтрация)

Получить: выборка (небольшая таблица)

Ключом, ключевым полем, называют поле в базе данных, содержащее значения, отличающие одни записи (объекты) от других. В качестве ключей могут выступать любые типы данных и любые поля таблицы – все определяется потребностями исследователей.

Критерий – это некоторое значение, с которым сравниваются значения всех записей, находящихся в ключевом поле. Критерий отбора назначает исследователь.

Фильтр – условия отбора записей по ключевым полям, в массиве сохраняются только те записи из общей базы данных, для которых значения в ключевых полях соответствуют условиям.

Формула условия, логического *сравнения значений с критериями* состоит из трех частей: **ключевое поле, *отношение*, значение критерия**. Выполнение условия позволяет отобрать из таблицы записи определенного качества и сформировать выборку для статистических расчетов. Например, формула `mm$spic == 'sar'` позволяет отобрать значения `'sar'` из колонки `spic` массива `mm` (см. стр. 62–63).

Возможны разные варианты формирования требований к выборке (все они будут подробно рассмотрены ниже):

1) в одной таблице относительно одного поля задать одно условие (выявление совпадений или превышение порога);

2) в одной таблице относительно одного поля задать два условия (отбор из диапазона значений);

3) в одной таблице относительно нескольких полей задать несколько условий;

4) для нескольких таблиц по нескольким полям задать несколько условий отбора записей.

Грубые ошибки при построении ключа

При сравнении ключа с критерием программа R всегда даст логический ответ – значения `TRUE` или `FALSE`. Однако необходимо соблюдать некоторые требования, чтобы ответ относился к содержанию полей, а не к их типу. *Типы данных для ключа и крите-*

рия должны быть *одинаковы*, для разных типов данных нужно использовать соответствующие им логические выражения.

Опасность № 1: использование вещественных чисел для идентификации записей (с оператором совпадения ==). Вещественное число имеет сложное компьютерное представление, включающее много цифр после запятой, которые могут быть не виды (в том числе возможна микроскопическая ошибка представления целого числа в младших разрядах). Из-за этого условие равенства, казалось бы, одинаковых чисел может не выполняться.

```
pi
[1] 3.141593

pi==3.141593
[1] FALSE

mm[20,] # длина особи 1507 равна 73.80000019073486
     n spic year season sex fer age yeseli line  w  lt  lc  lp
20 1507 sar 2002   6     m   ad  1  200206001 1  10 73.8 36 12.5

mm[20,11]==73.8
[1] FALSE
```

Рекомендация: ключи для отбора по *совпадению* значений должны быть либо целыми, либо текстовыми.

Опасность № 2: использование пробелов в текстовых константах. Эта проблема не связана с компьютером, но – с внимательностью оператора. Вписывая в качестве критерия текст, можно ошибиться с вводом пробелов, которые могут быть незаметны. В операции сравнения текстов выявляются первые, не совпадающие коды символов (упорядоченных по алфавиту). Поскольку `"c"` идет раньше `"s"`, ответ очевиден.

```
"sor">"clg"
[1] TRUE
```

При сравнении значений с пробелами легко ошибиться в позиционировании букв и тогда буква будет сравниваться с пробелом, который в кодовой таблице помещен раньше букв; в результате получаем неверный ответ.

```
"   sor">"  clg"
[1] FALSE
```

Рекомендация: включать в текстовое поле *очень простые и короткие* идентификаторы записей, не использовать пробелы.

Опасность № 3: разнообразие символов в ключевых полях типа текстовая константа. Сходство начертания символов латиницы и кириллицы приводит к неправильным результатам сравнения.

```
"сор"=="сор"
[1] FALSE
```

Рекомендация — ключевые поля наполнять строго *строчными символами на латинице*, избегая любых сопутствующих значков и специальных символов.

Отбор из одной таблицы: один ключ, один критерий

Формула условия отбора по одному критерию имеет самое простое строение: **ключевое поле – *отношение* – значение критерия**.

Левую позицию занимает имя ключевого поля. Критерий записывается справа как числовая или текстовая константа. Разнообразие логических выражений, проверяющих условие, указано в таблице.

Логическое отношение	Описание	Для сравнения каких типов данных
==	равно	целое, текстовое
!=	не равно	целое, текстовое
>	больше	целое, вещественное
>=	больше или равно	целое, вещественное
<	меньше	целое, вещественное
<=	меньше или равно	целое, вещественное

Условие отбора формует массив логических переменных, которые показывают, как соответствует данная запись критерию. Если для данной записи условие выполняется, результатом является **TRUE**, если не выполняется – **FALSE**. Проверим особей в колонке **spic** массива **mm** на принадлежность к виду обыкновенная бурозубка (**"sar"**):

```
mm$spic=="sar"
[1] TRUE FALSE  TRUE  FALSE  FALSE  TRUE  TRUE
```

Формула фильтра данных должна включать, во-первых, условие отбора данных, во-вторых, указание, с какими именно данными идет работа (т. е. имя массива): **имя массива [условие отбора, список нужных полей]**. Условие отбора порождает массив логических переменных, которые *играют роль индексов* отбираемых записей. Значение **TRUE** включает запись в выборку, значение **FALSE** — исключает: поскольку **TRUE** стоит на 1, 3, 6, 7-м местах, отбираются строки только с этими номерами.

```
mm<-read.csv("mamm.csv")
mm[mm$spic=="sar",]
    n  spic year season sex fer age yeseli line w   lt  lc  lp
2   977  sar 2001   6   f  ad   1 200106002  2 12.4 71.0 36 12.0
11  986  sar 2001   6   m  ad   1 200106002  2 10.4 74.0 39 12.0
20 1507  sar 2002   6   m  ad   1 200206001  1 10.2 73.8 36 12.5
26 1513  sar 2002   6      ad   1 200206001  1 11.0 73.0 40 12.0
```

Нужно заметить, что номера строк в выборке сохраняются от исходной базы данных. В примере в выборку перешли только строки 2, 11, 20, 26.

Условие отбора можно сохранить как отдельный объект и использовать как список индексов для отбора нужных записей из базы данных:

```
cs<-mm$spic=="sar"
mm[cs,]
    n  spic year season sex fer age yeseli line w   lt  lc  lp
2   977  sar 2001   6   f  ad   1 200106002  2 12.4 71.0 36 12.0
11  986  sar 2001   6   m  ad   1 200106002  2 10.4 74.0 39 12.0
20 1507  sar 2002   6   m  ad   1 200206001  1 10.2 73.8 36 12.5
26 1513  sar 2002   6      ad   1 200206001  1 11.0 73.0 40 12.0
```

Построенное выражение еще не является полноценным фильтром, поскольку содержит избыточное число полей, которое не попадает в обработку. Номера нужных полей указывают как вектор с их индексами на втором месте после запятой.

```
mm[mm$spic=="sar",c(10,11)]
      w   lt
2  12.4 71.0
11 10.4 74.0
20 10.2 73.8
26 11.0 73.0
```

Теперь полученную выборку можно использовать для статистических расчетов, например, корреляции (`cor()`), функция `round(...,2)` округляет выводимые значения; в примере – до двух знаков после запятой.

```
(res<-round(cor(mm[mm$spic=="sar",c(10,11)]),2))
      w    lt
w   1.00 -0.99
lt -0.99  1.00
```

Как можно видеть, в одну небольшую строку уместились и условия отбора записей, и условия отбора полей, и процедура расчета корреляций, и даже форматирования результатов для отчета. Чтобы оценить лаконичность (41 нажатие на клавиши) и красоту записей на языке R, предлагаем читателям задание – посчитать количество движений для получения такого же результата на листе Excel.

Отбор из одной таблицы: один ключ, два критерия

Более распространенная задача — отбор записей по значениям ключа из некоторого ограниченного диапазона. Например, отбор проб с координатами определенной географической области, отбор животных определенных размеров и пр. В этом случае составляется два условия (больше нижней критической границы и меньше верхней), которые объединяются пропозициональными союзами – логическим «И» или логическим «ИЛИ». Союз «И» означает, что отбираются только те записи, для которых выполняются оба условия, это жесткое требование. Союз «ИЛИ» означает, что отбираются те записи, для которых выполняются хотя бы какое-нибудь из условий, либо оба, это мягкое требование.

Союзы		Как выполняются условия:
&	логическое И	и одно, и другое; обязательно оба
|	логическое ИЛИ	или одно, или другое, или оба вместе

Отличие работы этих союзов хорошо видно в примере: жесткий критерий («и то, и другое») отбирает немного записей, мягкий (хоть что-нибудь) должен отобрать все. Поскольку в анализируемом поле присутствуют значения **NA**, для начала от них необходимо избавиться с помощью функции **na.omit()**.

```
mmw<-na.omit(mm[,c(2,10)])
c1.w<-mmw[,2]>20 & mmw[,2]<22
c1.w
 [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[10] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE  TRUE FALSE
[19] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE …
mmw[c1.w,]
   spic  w
13  cgl 21
17  cgl 21
```

Казалось бы, для одного поля нет смысла использовать мягкий критерий. Однако для отбора любых крайних (выскакивающих) значений он вполне пригоден.

```
(c2.w<-mmw[,2]<8 | mmw[,2]>30)
[1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE
[10] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[19] FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE …
```

```
mmw[c2.w,]
   spic  w
7   cgl 32
8   smi  6
22  cgl 55
28  cgl 35
29  cgl 39
31  mag 42
```

Отбор из одной таблицы: несколько ключей и критериев

Еще более типичная задача – отобрать записи по критериям разного вида, например, самцов бурозубок с массой больше 6 г. Для объединения условий используются те же логические союзы.

```
mm[mm$spic=='sar' & mm$sex=='m' & mm$w>6,]
     n  spic year season sex fer age  yeseli  line w  lt  lc   lp
11  986  sar 2001     6   m  ad   1  200106002   2 10 74.0 39 12.0
20 1507  sar 2002     6   m  ad   1  200206001   1 10  7.8 36 12.5
```

Объединение двух таблиц

Операцию объединения таблиц по ключу выполняет функция **merge()**. Смысл объединения состоит в том, чтобы из двух таблиц составить одну, из которой затем можно составить выборку с помощью приемов, рассмотренных выше. В примерах нам предстоит объединить таблицы **mamm.csv**, **line.csv** и **spic.csv**. Читаем таблицы из файлов и проверяем наличие записей.

```
tail(mm<-read.csv("mamm.csv"),1)
    n  spic year season sex fer age yeseli  line  w  lt lc  lp
62 1904 cgl 2003    6    f   juv NA 200306001  1  8.7 70 31 15.5
```

```
head(ln<-read.csv("line.csv"),3)
  nli  se ye   da  traday  biotop    yeseli
1   1  6 2001  19    150   meadow   200106001
2   2  6 2001  19    150   conifer  200106002
3   3  6 2001  19    150   mixed    200106003
```

```
head(sp<-read.csv("spic.csv"),2)
  spic         names
1  sar Sorex araneus
2  sis  Sorex isodon
```

Общими для первых двух таблиц являются поля год (`year`, `ye`), сезон (`season`, `se`), линия сезона (`line`, `nli`) давилок, а также поле `yeseli`, **заранее рассчитанное** по формуле **year*100000+season*1000+line**. Например, 2004*100000+6*1000+3 = 200406003. В таблице для линий каждая запись обрела уникальное значение этого ин-

декса. В таблице особей одинаковый индекс могли иметь несколько зверьков, попавших за один период отлова в эту линию. Объем таблиц получился разный: число записей в таблице для животных (**nrow(mm)**) – 62, для линий (**nrow(ln)**) – 12; это понятно, ведь в одну линию обычно попадалось несколько зверьков.

Общим полем для первой и третьей таблиц является **spic**, код названия вида. Объем этих таблиц также различается – в регионе зарегистрировано 19 видов, но в 2001–2003 гг. отловлено 62 особи 5 видов.

Связь таблиц без обобщения по простому ключу

Часто возникает задача добавить к каждой записи одной таблицы новые значения, взятые из второй таблицы, т. е. просто пополнить одну из таблиц дополнительной информацией. Эту операцию выполняет функция **merge()**. Например, вместо кодов видов в таблице хочется увидеть названия видов. Для этой цели необходимо добавить данные из таблицы с полными видовыми названиями к таблице с полной характеристикой зверьков, точнее говоря, к каждой записи первой таблицы добавить записи из второй таблицы, в которых ключи совпадают. Для примера возьмем не все поля, но только характеристики статуса особей (1–7). Важно обратить внимание на то, что все записи упорядочены по номеру **nn**, который соответствует датам отлова.

```
head(nmm[,c(1:7)])
  nn   n spic year season sex fer
1  1 976  cgl 2001      6   m  ad
2  2 977  sar 2001      6   f  ad
3  3 978  cgl 2001      6   m  sad
4  4 979  cgl 2001      6   m  sad
5  5 980  cgl 2001      6   m  ad
6  6 981  cgl 2001      6   m  ad
```

```
head(sp,3)
   spic      names
1  sar    Sorex araneus
2  sis    Sorex isodon
3  sca    Sorex caecutiens
```

Функция **merge()** берет две таблицы **nmm** и **sp**, и объединяет их по ключевому полю **spic** (**by='spic'**), т. е. к каждой записи из таблицы **nmm** подставляется запись из таблицы **sp**, *если значения в ключевом поле совпали*. Из полученной объединенной таблицы, которая теперь содержит не 14, а 15 полей, берем не все, но только семь первых полей и последнее (**[,c(1:7,15)]**) и заносим в массив **nmmsp**. На консоль выводим только последние три записи **tail(…, 3)** этого массива. Сравним его с исходной таблицей **nmm**.

```
tail(nmm[,c(1:7)],3)
   nn    n spic  year season sex fer
60 60 1902  cgl  2003    6    f  ad
61 61 1903  cgl  2003    6    m  ad
62 62 1904  cgl  2003    6    f  juv
```

```
tail(nmmsp<-merge(nmm,sp, by='spic')[,c(1:7,15)])
   spic nn    n year season sex  fer       names
57  sar 11  986 2001      6   m  ad    Sorex araneus
58  sar  2  977 2001      6   f  ad    Sorex araneus
59  sbe 35 1522 2002      6   f  ad Sicista betulina
60  sbe 33 1520 2002      6   f  ad Sicista betulina
61  sbe 30 1517 2002      6   f  ad Sicista betulina
62  smi  8  983 2001      6   f  ad    Sorex minutus
```

Как можно видеть, результирующая таблица **nmmsp** структурно перестроилась. Ключевое поле **spic** переместилось на первую позицию, добавилось поле **names** с полным названием вида. Порядок записей изменился и стал соответствовать алфавитной последовательности текста из ключевого поля **spic**.

Чтобы восстановить исходный порядок, используем функцию **order()**, которая возвращает номер места, на котором должна стоять запись из таблицы **nmmsp**, чтобы восстановилась их очередность по полю **nn** (от 1 до 62). Например, запись для малой бурозубки (**smi**), чье имя оказалось последним в алфавитном списке текущих отловов, в новой таблице занимает последнее место. На самом же деле, эта запись сделана раньше многих других и в упорядоченной по хронологии исходной таблице занимает восьмое место. Массив **oy** как раз и показывает, в частности, что запись № 62 из таблицы **nmmsp** в хронологическом порядке должна стоять на восьмом месте; результат запишем в **mmsp**.

```
(oy<-order(nmmsp$nn))
 [1] 1 58 3 4 5 6 7 62 9 10 57 12 13 14 2 16 17 18 19 56 8 22
[23] 23 11 25 55 27 15 29 61 54 32 60 21 59 36 24 38 26 40 28 42
[43] 30 31 45 20 34 35 49 37 51 39 52 41 53 43 44 46 33 47 48 50
```

```
tail(mmsp<-nmmsp[oy,],3)
  spic nn   n  year season sex  fer           names
47 cgl 60 1902 2003   6     f   ad    Clethrionomys glareolis
48 cgl 61 1903 2003   6     m   ad    Clethrionomys glareolis
50 cgl 62 1904 2003   6     f   juv   Clethrionomys glareolis
```

В этом разделе после объединения двух таблиц была получена новая таблица, включающая все исходные данные и которая теперь может поставлять полные выборки для дальнейшей обработки. Иногда связь между таблицами можно установить только после того, как выполнено определенное обобщение данных в одной из них, т. е. когда объединяются трансформированные таблицы.

Связь обобщенных таблиц

Часто в экологии возникает задача рассмотреть многолетнюю динамику численности видов на изучаемой территории. Для этого нужно рассчитать число зверьков, приходящихся на единицу промыслового усилия – число особей на 100 давилко-суток. Задача формулируется так: для определенного времени и места подсчитать число отловленных зверьков, подсчитать число давилко-суток и поделить первое число на второе.

Связь обобщенных таблиц без ключа

Для любой сортировки и любого обобщения информации требуются ключевые поля. Если в заголовке указано их отсутствие, значит, две таблицы должны быть тотально обобщены и выражены двумя числами (за рассматриваемый период) – общим количеством зверьков и общим объемом отловов – суммой давилко-суток.

```
100*nrow(mm)/sum(ln$traday)
[1] 3.444444
```

Общая средняя численность мелких млекопитающих весной 2001-2003 гг. в Гомсельге составила 3.4 экз./100 д-с.

Связь обобщенных таблиц по простому ключу

Теперь решим задачу оценки численности зверьков в разные годы. Эта цель назначает в качестве ключевого поля год, **`year`**. Сначала подсчитываем число зверьков и сумму давилко-суток по годам, т. е. составим два вектора по три значения для 2001, 2002, 2003 гг., затем делим один вектор на другой. Обобщение можно выполнить с помощью разных базовых функций. Функции обобщения сначала создают серию подмножество ячеек (векторов) с одинаковым значением ключа (**`year`**), затем выполняют с этими векторами операции обобщения; в нашем случае – подсчет числа значений, т. е. определяет длину подвекторов.

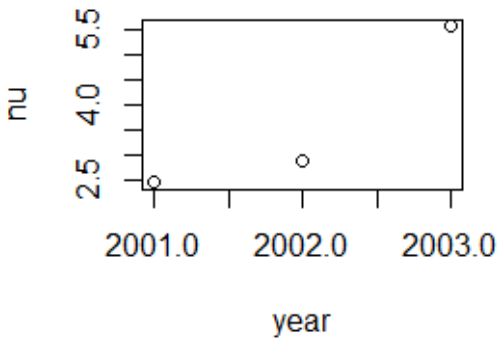
Отличие работы этих функций состоит в формировании объектов разных типов, что выясняется при рассмотрении их структуры (**`str()`**). Например, функция **`table()`** в большей степени рассчитана на последующее построение гистограммы. Нам же важно подготовить массив класса **`data.frame`** для объединения с другим таким же. Такой формат готовит функция **`aggregate()`**, однако поле **`year`** становится целочисленным. Если функцию **`table()`** перевести в формат **`data.frame`**, поле **`year`** станет фактором, что иногда бывает удобнее.

```
n0<-table(mm$year)
str(n0)
 'table' int [1:3(1d)] 11 26 25
 - attr(*, "dimnames")=List of 1
  ..$ : chr [1:3] "2001" "2002" "2003"
```

```
n1<-as.data.frame(table(mm$year))
str(n1)
'data.frame':   3 obs. of  2 variables:
 $ Var1: Factor w/ 3 levels "2001","2002",..: 1 2 3
 $ Freq: int  11 26 25
n2<-tapply(mm$year,mm$year,length)
str(n2)
 int [1:3(1d)] 11 26 25
 - attr(*, "dimnames")=List of 1
  ..$ : chr [1:3] "2001" "2002" "2003"
n3<-by(mm$year,mm$year,length)
str(n3)
'by' int [1:3(1d)] 11 26 25
 - attr(*, "dimnames")=List of 1
  ..$ mm$year: chr [1:3] "2001" "2002" "2003"
- attr(*, "call")= language by.default(data = mm$year, INDICES = mm$year, FUN =
length)
n4<-aggregate(mm$year,list(mm$year),length)
str(n4)
'data.frame':   3 obs. of  2 variables:
 $ Group.1: int  2001 2002 2003
 $ x      : int  11 26 25
```

Остановимся на функции **aggregate()**. Подсчитываем объемы давилко-суток (**d4**), объединяем таблицы (**nd4**), рассчитываем отношение между столбцами и добавляем отношение к новой таблице (**nd4r**), меняем имена полей (**names(nd4r)**), строим диаграмму (**plot()**). Как видно, в 2003 г. наблюдался рост численности зверьков.

```
d4<-aggregate(ln$traday,list(ln$ye),sum)
nd4<-merge(n4,d4,by='Group.1')
nd4r<-data.frame(nd4,100*nd4[,2]/nd4[,3])
names(nd4r)<-c('year','ex','ds','nu')
nd4r
  year ex  ds       nu
1 2001 11 450 2.444444
2 2002 26 900 2.888889
3 2003 25 450 5.555556
plot(nd4r[,c(1,4)])
```



Справиться с задачей удалось достаточно просто еще и потому, что при объединении таблиц использован метод «один-к-одному», т. е. каждая запись одной таблицы получила запись из другой таблицы. Однако иногда таблицы имеют неодинаковый размер.

Связь обобщенных таблиц по составному ключу

Возможна более тонкая постановка задачи, чем в предыдущем разделе, например, оценка численности зверьков в разных биотопах, или в разных линиях, или межгодовая динамика в разных биотопах и пр. В этой ситуации для объединения таблиц потребуется выполнить несколько шагов. Нужно определиться с условиями подбора, подсчитать число зверьков того или иного вида, попавших в отдельные линии, далее – таблицы по зверькам и объемам отлова и в заключение выполнить требуемое обобщение, расчет численности и вывести иллюстрацию.

Таблицы для животных и для ловушек не комплиментарны друг другу, поскольку на одну запись из таблицы линий приходится несколько записей из таблицы животных. Поэтому перед объединением нужно подсчитать число особей, попавших в отдельную линию ловушек. Нужны четкие уникальные идентификаторы линии ловушек в обеих таблицах (**mm** и **ln**). Казалось бы, таким идентификатором должен быть сквозной номер линии. Однако на практике сделать это невозможно, поскольку в экологии традиционно точкам отбора проб даются постоянные номера (приведенные линии 1, 2 и 3 привязаны к территории с 1993 г.), которые повторяются от отлова к отлову. Кроме того, отлов могут вести несколько групп в новых местах и номера их линий могут совпадать. Хорошим идентификатором линий (и ключевым полем) может служить сложный ключ, составленный из трех простых **yeseli= year*100000+season*1000+line.** Кратко запишем порядок предстоящей работы:

Дано: таблицы по числу зверьков и объемам отлова
- уточняем статус зверьков (вид, пол, зрелость)
- для значений `yeseli` определяем сумму зверьков со статусом `nm`
- связываем таблицы **nc** и **ln** по общему ключу `yeseli` (таблица **lnc**)
- уточняем критерии (ключи) обобщения записей таблицы **lnc**
- обобщаем записи таблицы **lnc**
- делим число зверьков на объём отловов

Получить: график динамики относительной численности

Рассчитаем межгодовую (**year**) динамику численности рыжей полевки (**cgl**). Отбираем из базы только рыжих полевок (в таблицу **cgl**).

```
head(cgl<-mm[mm$spic=='cgl',],3)
    n spic year season sex fer age   yeseli line w   lt lc lp
1 976  cgl 2001      6   m  ad   9 200106002  2 24.0 95 40 16
3 978  cgl 2001      6   m sad   2 200106002  2 24.4 91 39 16
4 979  cgl 2001      6   m sad   2 200106002  2 25.0 99 43 17
```

Выполняем подсчет зверьков этого вида для каждой линии по ключу **yeseli**.

```
nc<-aggregate(cgl$yeseli,list(cgl$yeseli),length)
names(nc)<-c('yeseli','nm') # меняем имена полей
nc
      yeseli nm
1  200106001  1
2  200106002  7
3  200205001  2
4  200205002  5
5  200205003  1
6  200206001  4
7  200206002  5
8  200206003  3
9  200306001  9
10 200306002 11
11 200306003  5
```

Эту таблицу по ключу **yeseli** нужно присоединить к таблице **ln** с объемами отлова.

```
ln <- read.csv("line.csv")
ln
   nli se   ye da traday  biotop    yeseli
1    1  6 2001 19    150  meadow 200106001
2    2  6 2001 19    150 conifer 200106002
3    3  6 2001 19    150   mixed 200106003
4    1  5 2002  4    150  meadow 200205001
5    2  5 2002  4    150 conifer 200205002
6    3  5 2002  4    150   mixed 200205003
7    1  6 2002 24    150  meadow 200206001
8    2  6 2002 24    150 conifer 200206002
9    3  6 2002 24    150   mixed 200206003
10   1  6 2003 19    150  meadow 200306001
11   2  6 2003 19    150 conifer 200306002
12   3  6 2003 19    150   mixed 200306003
```

Сравнивая таблицы, нетрудно заметить, что они имеют разное количество строк; видимо, в линию 200106003 никто не попался. Если связывать таблицы методом «один-к-одному», результирующая таблица будет иметь 11 записей, т. е. отловы в линию 200106003 как бы не проводились. Однако это не так, отловы проводились, но зверьки не попались, т. е. численность должна получить значение 0. Если этого не сделать, обобщения будут содержать ошибку – неоправданно увеличивать численность. Чтобы ее не допустить, необходимо связывать таблицы, специально указывая, что результат должен включить все записи. Это достигается с помощью аргумента **all=TRUE** (по умолчанию **all=FALSE**).

```
(lnc<-merge(ln,nc,by='yeseli',all=TRUE))
      yeseli nli se   ye da traday  biotop nc
1  200106001   1  6 2001 19    150  meadow  1
2  200106002   2  6 2001 19    150 conifer  7
3  200106003   3  6 2001 19    150   mixed NA
4  200205001   1  5 2002  4    150  meadow  2
5  200205002   2  5 2002  4    150 conifer  5
6  200205003   3  5 2002  4    150   mixed  1
7  200206001   1  6 2002 24    150  meadow  4
8  200206002   2  6 2002 24    150 conifer  5
9  200206003   3  6 2002 24    150   mixed  3
10 200306001   1  6 2003 19    150  meadow  9
11 200306002   2  6 2003 19    150 conifer 11
12 200306003   3  6 2003 19    150   mixed  5
```

Связь выполнена правильно, но численность в линии `200106003` получила значение `NA`, которое следует заменить на 0. Для начала нужно найти, в какой ячейке находится неопределенное значение. Простые логические операторы с этой константой не работают.

```
lnc[,8]==NA
[1] NA NA NA NA NA NA NA NA NA NA NA NA
```

Необходимо воспользоваться специальной функцией определения типа данных.

```
is.na(lnc[,8])
[1] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE …
```

Найдя нужное значение, заменяем его на ноль:

```
lnc[is.na(lnc[,8]),8] <- 0
lnc
      yeseli nli se   ye da traday  biotop nc
1  200106001   1  6 2001 19    150  meadow  1
2  200106002   2  6 2001 19    150 conifer  7
3  200106003   3  6 2001 19    150   mixed  0
4  200205001   1  5 2002  4    150  meadow  2
5  200205002   2  5 2002  4    150 conifer  5
6  200205003   3  5 2002  4    150   mixed  1
7  200206001   1  6 2002 24    150  meadow  4
8  200206002   2  6 2002 24    150 conifer  5
9  200206003   3  6 2002 24    150   mixed  3
10 200306001   1  6 2003 19    150  meadow  9
11 200306002   2  6 2003 19    150 conifer 11
12 200306003   3  6 2003 19    150   mixed  5
```

Теперь можно выполнить обобщения данных по разным ключам (суммирование зверьков и давилко-суток) и рассчитать относительные оценки:

1) динамика численности по годам – ключ **lnc$ye**

```
(ncy<-aggregate(lnc[,c(6,8)],list(lnc$ye),sum))
  Group.1 traday nc
1    2001    450  8
2    2002    900 20
3    2003    450 25
```

```
ncym<-data.frame(ncy,round(100*ncy[,3]/ncy[,2],2))
names(ncym)<-c('year','traday','ex','numy')
ncym
  year traday ex numy
1 2001    450  8 1.78
2 2002    900 20 2.22
3 2003    450 25 5.56
```

2) отличия численности по биотопам – ключ **lnc$biotop**

```
(ncb<-aggregate(lnc[,c(6,8)],list(lnc$biotop),sum))
  Group.1 traday nc
1 conifer    600 28
2  meadow    600 16
3   mixed    600  9
```

```
ncbm<-data.frame(ncb,round(100*ncb[,3]/ncb[,2],2))
names(ncbm)<-c('biotop','traday','ex','numb')
ncbm
   biotop traday ex numb
1 conifer    600 28 4.67
2  meadow    600 16 2.67
3   mixed    600  9 1.50
```

3) динамика численности по биотопам

```
ncyb<-aggregate(lnc[,c(6,8)],list(lnc$ye,lnc$biotop),sum)
ncybm<-data.frame(ncyb,round(100*ncyb[,4]/ncyb[,3],2))
names(ncybm)<-c('year','biotop','traday','ex','numb')
ncybm
  year  biotop traday ex numb
1 2001 conifer    150  7 4.67
2 2002 conifer    300 10 3.33
3 2003 conifer    150 11 7.33
4 2001  meadow    150  1 0.67
5 2002  meadow    300  6 2.00
6 2003  meadow    150  9 6.00
7 2001   mixed    150  0 0.00
8 2002   mixed    300  4 1.33
9 2003   mixed    150  5 3.33
```

Ликвидация пропусков в выборках

Никакие натурные наблюдения не обходятся без того, чтобы в базе данных не пропали какие-либо записи. В таблице начинают зиять пробелы, в среде языка R обретающие значения **NA**. Что с ними делать? Никакая статистическая функция R не будет работать с неопределенными данными, результат всегда будет такой же. Пробелы нужно ликвидировать. Сразу нужно оговориться – ликвидировать только в выборках; в исходной базе данных пробелы в данных следует оставить. Исправленные пробелы пропадают из поля зрения и навсегда остаются в базе, хотя в будущем могут появиться более эффективные способы исправления пустот. В наших данных у некоторых зверьков отсутствуют промеры (сильно погрызенные трупики), значение пола (съедены внутренние органы), отсутствуют данные по зрелости и возрасту. Обнаружить пробелы в таблицах позволяют функции **is.na()**, **anyNA()**, **complete.cases()**, **na.action(na.omit()).**

Есть два генеральных пути для ликвидации пробелов в выборках. Данные с пропусками можно исключить из анализа. Этому служат функция **na.omit()**, которая может удалить элементы вектора или целиком ряды и столбцы таблицы, запись массив [-список, -список], аргумент **na.rm=TRUE**, встроенный в большинство статистических функций, который не берет в расчет ячейки или ряды с пробелами. Главное возражение против этого подхода состоит не только в том, что в результате уменьшается объем выборок, а то, что пропадает вся косвенная информация об объекте, заключенная в других его характеристиках.

Другой путь – подставить на места пустот некие более или менее правдоподобные значения, заменяющие отсутствующие. В этом случае объем выборок сохраняется. Однако исходя из общих статистических соображений, новые значения, как минимум, не должны существенно искажать статистическое свойства распределения, т. е. смещать среднюю и изменять дисперсию. Кроме того, желательно, чтобы новое значение было как-то индивидуализировано, соответствовало по степени выраженности статусу объекта с пропусками. В практике статистического анализа разработано множество процедур восстановления данных (см. библиографию). Сразу оговоримся, что в литературе не рекомендуют для подстановки использовать среднее арифметическое или медиану, которые неизбежно увеличивают эксцесс и снижают дисперсию. Рассмотрим два достаточно простых работающих метода: это простая регрессия и подстановка с подбором внутри групп. Опираясь на здравый смысл, необходимо к пропускам разных характеристик объектов подходить с разными мерками, чтобы понять, нужно ли вообще данные восстанавливать, возможно ли это сделать теоретически и какую именно методику применить.

Удаление серийных пробелов

Цель статистической обработки промеров обычно состоит либо в том, чтобы, получив параметры для выборок разных групп, сравнить их на предмет биологических отличий (дисперсионный анализ), либо в том, чтобы оценить зависимость одних промеров от других (регрессионный анализ). Обычно оба вида анализа соседствуют. Если пробелы обнаруживаются у одной особи одновременно в серии промеров, значит, их восстановление по одинаковой методике породит близкие значения, которые неизбежно усилят корреляцию (а подстановка средних еще и снизит дисперсию). Вывод очевиден: серийные пробелы в одной записи – это показание к полному удалению записи из анализа.

Рассмотрим данные многочисленного вида, рыжей полевки, подготовив отдельный массив (**mcgl**).

```
mm<-read.csv("mamm.csv")
head(mcgl<-mm[mm$spic=='cgl',],2)
    n spic year season sex fer age    yeseli line    w lt lc lp
1 976  cgl 2001      6   m  ad   9 200106002    2 24.0 95 40 16
3 978  cgl 2001      6   m sad   2 200106002    2 24.4 91 39 16
```

После изъятия части данных нумерация записей нарушается, для дальнейшего анализа восстанавливаем последовательность номеров записей.

```
(rownames(mcgl)<-c(1:nrow(mcgl)))
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
[21] 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
[41] 41 42 43 44 45 46 47 48 49 50 51 52 53
```

Рассматривая наши данные, видно, что некоторые погрызенные особи имеют пробелы в промерах длины тела и хвоста.

```
mcgl[is.na(mcgl['lt']& mcgl['lc']),]
      n spic year season sex fer age  yeseli  line  w  lt lc lp
35 1877  cgl 2003     6   m      NA 200306003   3 13.4 NA NA NA
36 1878  cgl 2003     6   m      NA 200306001   1 23.4 NA NA NA
```

Строчки 35 и 36 нужно удалить из выборки. «Вытащить» эти номера из списка можно разными способами.

```
(exr<-as.integer(rownames(mcgl[is.na(mcgl['lt']&mcgl['lc']),])))
[1] 35 36
(rem<-c(as.integer(na.action(na.omit(mcgl['lt']& mcgl['lc'])))))
[1] 35 36
```

Удаляем ненужные строки из массива, сохранив его с новым именем (**mcg**), и обновляем номера строк.

```
mcgl[34:37,]
     n   spic year season sex fer age  yeseli line w  lt lc lp
34 1876  cgl 2003   6     f      NA 200306001  1 28.5 93 42 18
35 1877  cgl 2003   6     m      NA 200306003  3 13.4 NA NA NA
36 1878  cgl 2003   6     m      NA 200306001  1 23.4 NA NA NA
37 1879  cgl 2003   6     m      NA 200306003  3 13.4 75 32 17
```

```
mcg<-mcgl[-rem,]
mcg[34:37,]
     n spic year season sex fer age    yeseli line   w  lt lc lp
34 1876  cgl 2003     6   f      NA 200306001    1 28.5 93 42 18
37 1879  cgl 2003     6   m      NA 200306003    3 13.4 75 32 17
38 1880  cgl 2003     6   m      NA 200306001    1 16.7 84 41 18
39 1882  cgl 2003     6   m  ad  NA 200306002    2 18.1 86 36 18
```

```
(rownames(mcg)<-c(1:nrow(mcg)))
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
[24] 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43
[44] 44 45 4647 48 49 50 51
```

Заполнение пробелов в выборках методом простой регрессии

Восстановим пробелы по массе тела, ориентируясь на размеры тела. Как видно, в массиве **mcg** есть пробелы в поле массы тела **w**, что связано с частичным, но не существенным нарушением сохранности трупов. Для начала посмотрим, где в таблице находятся пробелы (условия **is.na** – «это пробел?»).

```
mcg[is.na(mcg$w),]
      n spic year season sex fer age  yeseli line w lt  lc  lp
38 1883  cgl 2003     6   f  ad  NA 200306001  1 NA 101 50 17.9
40 1885  cgl 2003     6   f  ad  NA 200306002  2 NA  88 36 17.0
41 1886  cgl 2003     6          NA 200306001  1 NA  84 42 17.1
43 1888  cgl 2003     6   f  ad  NA 200306003  3 NA  95 52 17.1
45 1890  cgl 2003     6   f juv  NA 200306003  3 NA  77 34 16.0
46 1891  cgl 2003     6   f juv  NA 200306001  1 NA  83 37 17.0
47 1892  cgl 2003     6     juv  NA 200306001  1 NA  65 30 16.2
```

Далее определим номера рядов с пробелами.

```
(nn<-as.integer(na.action(na.omit(mcg[,10]))))
[1] 38 40 41 43 45 46 47
```

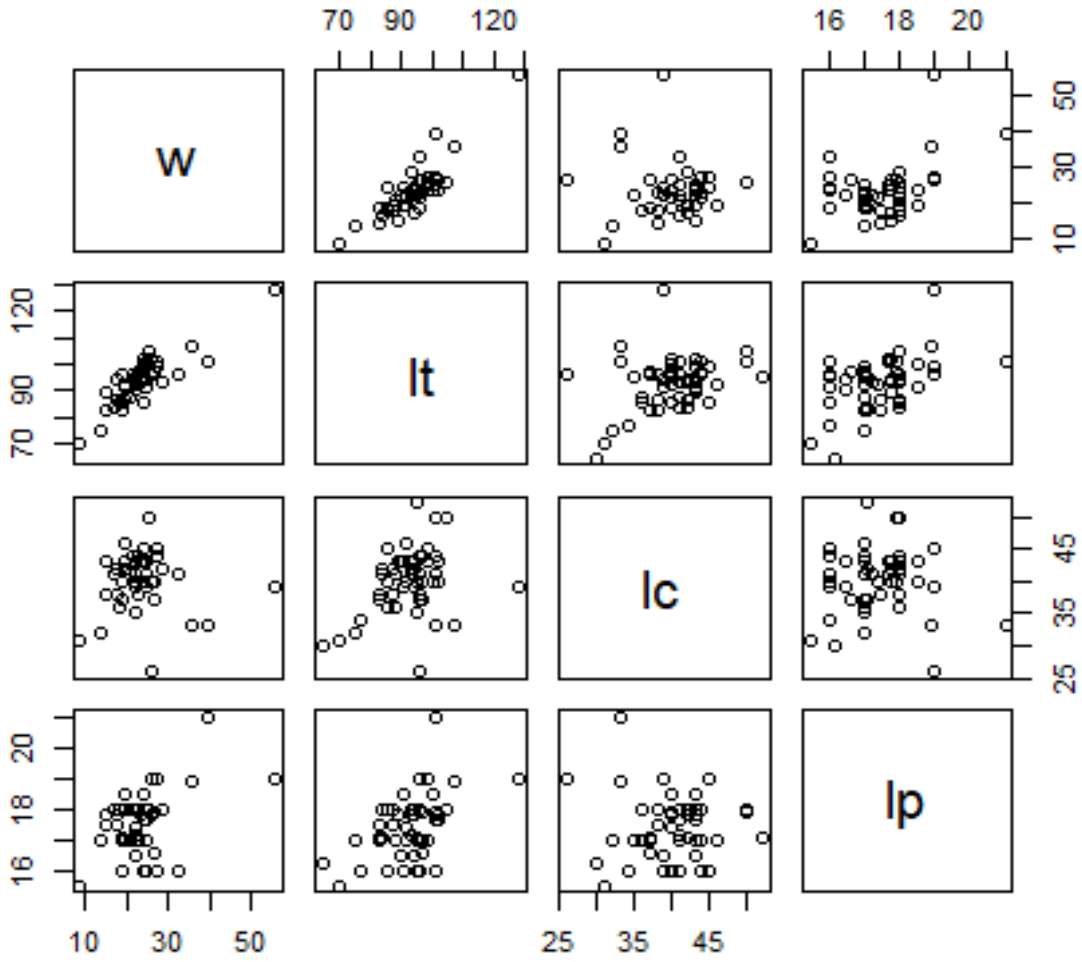
Здесь видна «матрешка» программы R. Поле 10 с данными массы рассматривается на предмет ликвидации пробелов с помощью функции **na.omit()**, которая должна возвращать (сохранять) только ряды *без пробелов*. Однако она вложена в функцию **na.action()**, которая возвращает номера рядов, требующие исключения, т. е. которые содержат NA. Однако результирующий массив имеет сложную структуру и непосредственно не годится в качестве индексов рядов, поэтому с помощью функции

as.integer() этот список превращен в целочисленный вектор с номерами рядов, содержащих пробелы. Теперь у нас есть ключ для манипуляции с пробелами.

Масса тела напрямую связана с размерами тела, т. е. с полученными нами промерами. Это позволяет использовать метод регрессии, включающий два этапа. Используя *полные данные*, рассчитаем уравнение зависимости массы тела особи от ее размеров, затем заполним пробелы прогнозами по этому уравнению.

Для начала необходимо понять, какой из промеров обладает наилучшими прогностическими способностями (возможен прогноз для промеров и с помощью уравнения многомерной регрессии, но мы его рассматривать не будем). Построим диаграмму зависимостей массы рыжей полевки (`cgl`) ото всех промеров.

```
plot(mcg[,10:13])
```



Оказалось, что только длина тела (`lt`) довольно строго связана с массой (`w`) особи.

Подготовим массивы без пробелов для массы (`y`) и длины (`x`) длины тела, исключая из рассмотрения (**-nn**) полученный ранее список рядов с пробелами.

```
(y<-mcg[-nn,10])
 [1] 24.0 24.4 25.0 24.3 23.0 32.7 18.7 27.6 23.8 21.7 22.0 14.9
[13] 19.3 21.824.8 26.8 17.4 55.8 26.3 22.4 23.9 24.7 35.8 39.5
[25] 18.7 22.9 20.7 26.227.5 19.6 17.8 23.4 22.0 28.5 13.4 16.7
[37] 18.1 26.6 18.9 22.1 25.6 14.7 19.2 8.7
(x<-mcg[-nn,11])
 [1]  95  91 99  86 97  96  86 101 102 96 95 89 92 92 101 97 94
[18] 128 102 94 100 94 107 101  96  96 88 96 99 87 87  93 90 93
[35]  75  84 86  97 83  93 105  83  91 70
```

Далее подготовим массив значений длины тела **newx** для прогноза, они взяты у особей с пропусками массы. Важно соблюсти именно рассмотренный формат данных.

```
(newx<-data.frame(x=mcg[nn,11]))
    x
1 101
2  88
3  84
4  95
5  77
6  83
7  65
```

Теперь в одной строке записываем программу расчета уравнения линейной регрессии между массой и длиной, и заполнение пробелов в данных для рыжей полевки округленными прогнозами по регрессии.

```
(mcg[nn,10] <- round(predict(lm(y~x),newdata=newx),1))
   1    2    3    4    5    6    7
28.6 19.3 16.5 24.3 11.6 15.8  3.0
mcg[nn,]
   n   spic year season sex fer age  yeseli line  w  lt  lc  lp
38 1883 cgl 2003   6    f  ad NA 200306001   1  28.6 101 50 17.9
40 1885 cgl 2003   6    f  ad NA 200306002   2  19.3  88 36 17.0
41 1886 cgl 2003   6         NA 200306001   1  16.5  84 42 17.1
43 1888 cgl 2003   6    f  ad NA 200306003   3  24.3  95 52 17.1
45 1890 cgl 2003   6    f juv NA 200306003   3  11.6  77 34 16.0
46 1891 cgl 2003   6    f juv NA 200306001   1  15.8  83 37 17.0
47 1892 cgl 2003   6      juv NA 200306001   1   3.0  65 30 16.2
```

Заполнение пробелов методом подбора внутри групп

Смысл процедуры состоит в том, чтобы для особей с пропусками в данных подобрать группу похожих на них объектов и взять от них недостающие промеры (случайным образом).

Выявляем из базы выборку рыжих полевок с пробелами в поле «масса тела»:

```
mcg<-mmm[mmm$spic=='cgl',]
mcg[is.na(mcg$w),]
     n  spic year season sex fer age   yeseli  line w   lt lc  lp
49 1883  cgl 2003      6   f  ad  NA 200306001    1 NA 101 50 17.9
51 1885  cgl 2003      6   f  ad  NA 200306002    2 NA  88 36 17.0
52 1886  cgl 2003      6          NA 200306001    1 NA  84 42 17.1
54 1888  cgl 2003      6   f  ad  NA 200306003    3 NA  95 52 17.1
56 1890  cgl 2003      6   f juv  NA 200306003    3 NA  77 34 16.0
57 1891  cgl 2003      6   f juv  NA 200306001    1 NA  83 37 17.0
58 1892  cgl 2003      6     juv  NA 200306001    1 NA  65 30 16.2
```

Особи `1883`, `1885`, `1888` – взрослые (`ad`) самки (`f`), особь `1886`, видимо, такая же. Рассмотрим только эту группу.

Из всей выборки взрослых самок возьмем значения массы тела без пробелов:

```
na.omit(mc[mcg$sex=='f'&mcg$fer=='ad',10])
 [1] 32.7 22.0 14.9 19.3 26.8 55.8 24.7 35.8 39.5 20.7 17.8 25.6 14.7
attr(,"na.action")
[1] 12 13 14
attr(,"class")
[1] "omit"
```

Далее случайным образом извлечем из нее 4 значения

```
(wcg<-sample(na.omit(mc[mcg$fer=='ad',10]),4))
[1]  19.2 14.7 21.8 24.7
```

и заполним этими значениями пустоты для взрослых самок рыжих полевок

```
mcg[is.na(mcg$w),][1:4,10]<-wcg
```

Результат достигнут, но на этом примере стоит показать, в чем преимущество предложенных замен. Для этого построим диаграмму с исходными данными, с прогнозом по регрессии, с выборкой из значений у похожих особей и средними значениями.

Разместим особей в порядке возрастания длины тела, но на диаграмму нанесем только данные по их массе.

```
mcg <- mcgl[-rem,]
nmcg <- round(predict(lm(y~x),newdata=newx),1)
nnn<-1:nrow(mcg)
plot(nnn,mcg[order(mcg[,11]),10],type='l',lwd=1,ylab='w')
(pn<-as.integer(na.action(na.omit(mcg[order(mcg[,11]),10]))))
[1]  1  4  6  9 16 31 46
points(pn,sort(predict(lm(y~x),newdata=newx)),pch=4,lwd=2)
mw<-rep(mean(mcg[,10],na.rm=TRUE),7)
points(pn,mw, fg=1,pch=21,bg=1,cex=1)
points(pn[4:7],wcg)
```

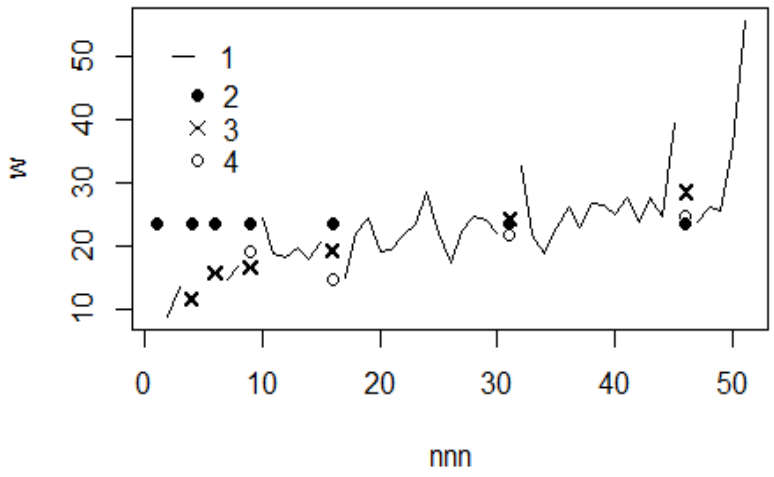


Диаграмма хорошо отражает рост массы по мере увеличения длины тела (1), но с довольно большой случайной изменчивостью. Анализ диаграммы позволяет сделать несколько выводов. Очевидно, что две крайние особи с массой 3 и 55 г не относятся к виду «рыжая полевка», это бурозубка и темная полевка (после исправления этой ошибки необходимо заново выполнить пересчет пропуска значений массы). Хорошо видно, что прогноз для пустот по регрессии (3) в большинстве случаев находится ближе к эмпирическим данным, чем средняя для всего ряда (2). Это показывает, что среднюю нельзя использовать для подстановки. Правая регрессионная точка пропуска находится дальше от ожидаемой массы, что, очевидно, связано с использованием очень высокого неправильного значения массы самой крупной полевки. В варианте случайной подстановки для взрослых самок точки укладываются в диапазон варьирования, но без трендов (4).

РЕШЕНИЕ ТИПИЧНЫХ БИОМЕТРИЧЕСКИХ ЗАДАЧ

В этой главе собраны примеры решения обычных задач, стоящих перед биологами, когда требуется для определенной биологической задачи подобрать адекватный статистический метод, применить его к имеющимся данным, получить статистический и биологический выводы, построить простую, но выразительную иллюстрацию. В числе таких методов – оценка параметров эмпирической выборки и распределения, статистическое сравнение оценок параметров и распределений, поиск зависимости между переменными, оценка действия факторов с помощью дисперсионного анализа, обобщение данных в форме суммативных индексов – главных компонент.

Описательная статистика

Когда параметры выборок нужны для создания иллюстраций или включения в таблицы, можно воспользоваться набором простых стандартных функций. Сначала читаем данные и извлекаем значения, относящиеся к одному виду, например `sor`.

```
head(mm<-read.csv("mamm.csv"))
    n spic year season sex fer age    yeseli line    w lt lc lp
1 976  cgl 2001      6   m  ad   9 200106002    2 24.0 95 40 16
2 977  sar 2001      6   f  ad   1 200106002    2 12.4 71 36 12
3 978  cgl 2001      6   m sad   2 200106002    2 24.4 91 39 16
4 979  cgl 2001      6   m sad   2 200106002    2 25.0 99 43 17
5 980  cgl 2001      6   m  ad   9 200106002    2 24.3 86 45 16
6 981  cgl 2001      6   m  ad   9 200106002    2 23.0 97 44 17
```

```
(m.sar<-mm[mm$spic=='sar',10:13])
      w   lt lc   lp
2  12.4 71.0 36 12.0
11 10.4 74.0 39 12.0
20 10.2 73.8 36 12.5
26 11.0 73.0 40 12.0
```

Для общего представления данных достаточно функции **`summary()`**, которая показывает пределы варьирования, квартили 25 % и 75 %, медиану и среднюю.

```
summary(mm[mm$spic=='sar',10:13])
       w               lt              lc              lp
 Min.   :10.20   Min.   :71.00   Min.   :36.00   Min.   :12.00
 1st Qu.:10.35   1st Qu.:72.50   1st Qu.:36.00   1st Qu.:12.00
 Median :10.70   Median :73.40   Median :37.50   Median :12.00
 Mean   :11.00   Mean   :72.95   Mean   :37.75   Mean   :12.12
 3rd Qu.:11.35   3rd Qu.:73.85   3rd Qu.:39.25   3rd Qu.:12.12
 Max.   :12.40   Max.   :74.00   Max.   :40.00   Max.   :12.50
```

К сожалению, эта функция выдает результаты в текстовом формате, не пригодном для расчета, а также не содержит значений дисперсии. Этот и другие параметры можно рассчитать с помощью функций **`sd()`**, **`mean()`**, **`median()`**, **`min()`**, **`max()`**, **`quantile()`** и др.

```
sd(mm[mm$spic=='sar',10])
[1] 0.993311
```

Есть функции, позволяющие рассчитать статистические параметры сразу для нескольких групп, если их представители снабжены индексом своей группы. В нашем примере в поле `spic` указана принадлежность каждой особи к одному из четырех видов. Функции векторных операций позволяют для любого отдельного признака рассчитать любой статистический параметр. Для колонок, содержащих пробелы `NA`, нужно указать условие их игнорирования **`na.rm=TRUE`**. Для двух видов рассчитать стандартное отклонение не удалось, поскольку выборки содержали лишь по 1 значению, на что указывают результаты подсчёта объёма выборок (**`length`**).

```
tapply(mm$w,mm$spic,sd,na.rm=TRUE)
     cgl      mag      sar      sbe      smi
7.449864       NA 0.993311 2.154840       NA
```

```
tapply(mm$w,mm$spic,length)
cgl  mag  sar  sbe  smi
 53    1    4    3    1
```

или

```
aggregate(mm$w,list(mm$spic),length)
  Group.1  x
1     cgl 53
2     mag  1
3     sar  4
4     sbe  3
5     smi  1
```

Используя эти функции, можно сразу рассчитать ошибки средних (m=s/n^0.5).

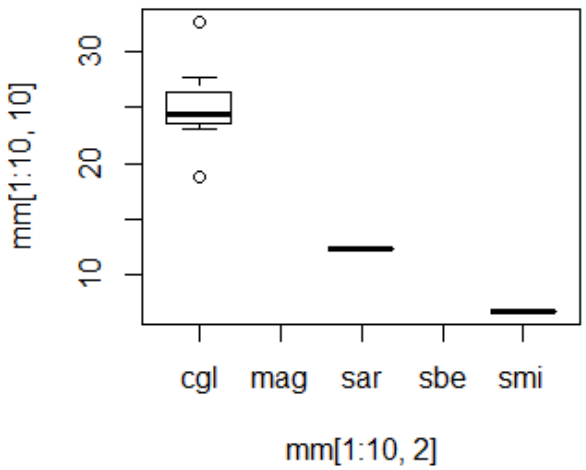
```
tapply(mm$w,mm$spic,sd,na.rm=TRUE)/sqrt(tapply(mm$w,mm$spic, length))
     cgl  mag      sar       sbe    smi
1.0233175   NA 0.4966555 1.2440972   NA
```

Наглядное представление о параметрах дает диаграмма типа **`boxplot()`**; в списке аргументов нужно указать, что числовая характеристика, масса тела (**`mm[1:10,10]`**), распределяется (знак тильда **~**) по видовым группам (**`mm[1:10,2]`**). Результаты обработки выборок можно вывести в окно консоли, используя аргумент **`plot=FALSE`**.

```
boxplot(mm[1:10,10]~mm[1:10,2])
```

```
boxplot(mm[1:10,10]~mm[1:10,2], plot=FALSE)
$stats
      [,1]  [,2] [,3] [,4] [,5]
[1,] 23.00   NA  12.4   NA  6.7
[2,] 23.50   NA  12.4   NA  6.7
[3,] 24.35   NA  12.4   NA  6.7
[4,] 26.30   NA  12.4   NA  6.7
[5,] 27.60   NA  12.4   NA  6.7
```

```
$n
[1] 8 0 1 0 1
$conf
         [,1] [,2] [,3] [,4] [,5]
[1,] 22.78588   NA 12.4   NA  6.7
[2,] 25.91412   NA 12.4   NA  6.7
$out
[1] 32.7 18.7
$group
[1] 1 1
$names
[1] "cgl" "mag" "sar" "sbe" "smi"
```



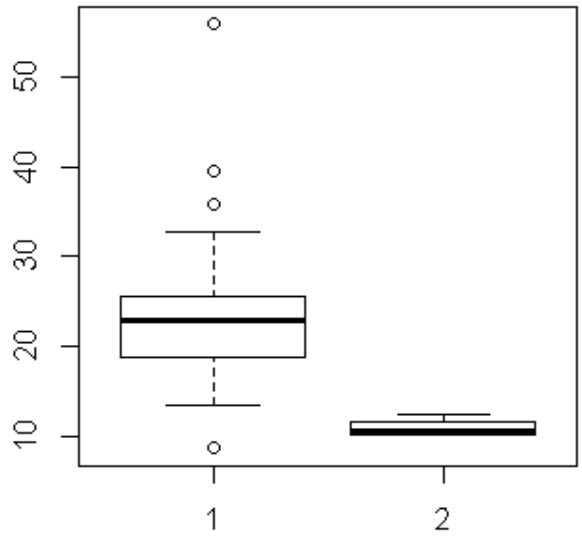
В столбцах блока `stat` представлены основные статистики для выборок, перечисленных в блоке `names` – `cgl` (`[,1]`), `mag` (`[,2]`), `sar` (`[,3]`), `sbe` (`[,4]`) и `smi` (`[,5]`): минимальное значение `[1,]`, первый квартиль `[2,]`, медиана `[3,]`, третий квартиль `[4,]` и максимальное значение `[5,]`, далее идут `n` – объем выборок, `conf` – верхняя и нижняя границы доверительного интервала для медианы («усы»); `out` – «выбросы», значения выборок за пределами «усов»; `group` – указание на выборку (порядковый номер столбца) с «выбросами»; `names` – имена выборок (столбцов).

Сравнение выборок

Выборочные средние арифметические можно сравнить по критерию Стьюдента (**`t.test()`**), а стандартные отклонения – по критерию Фишера (**`var.test()`**). Эти параметрические критерии используются, когда есть основание считать, что изучаемые признаки имеют нормальное распределение (что характерно для массы тела). Для проверки этого служит критерий Шапиро – Уилкса (**`shapiro.test()`**). Если обнаружилось отличие от нормальности, выборки можно сравнить в целом, с помощью непараметрического критерия Вилкоксона – Манна – Уитни (**`wilcox.test()`**), который показывает значимые *смещения* выборок относительно друг друга. Варианты в выборках следует предварительно упорядочить от меньшего к большему значению (**`sort()`**) и для визуализации смещений построить диаграмму «ящик с усами» (**`boxplot()`**). Сравним полевок (**`cg`**) и бурозубок (**`sa`**) по массе тела (**`w`** – 10-й столбец массива **`mm`**)

```
mm<-read.csv("mamm.csv")
(cg<-sort(mm[mm$spic=='cgl',10]))
[1]  8.7 13.4 13.4 14.7 14.9 16.7 17.4 17.8 18.1 18.7 18.7
[12] 18.9 19.2 19.3 19.6 20.7 21.7 21.8 22.0 22.0 22.1 22.4
[23] 22.9 23.0 23.4 23.4 23.8 23.9 24.0 24.3 24.4 24.7 24.8
[34] 25.0 25.6 26.2 26.3 26.6 26.8 27.5 27.6 28.5 32.7 35.8
[45] 39.5 55.8
(sa<-sort(mm[mm$spic=='sar',10]))
[1] 10.2 10.4 11.0 12.4
```

```
boxplot(cg,sa)
```




```
shapiro.test(cg); shapiro.test(sa)
        Shapiro-Wilk normality test
data:  cg
W = 0.84628, p-value = 2.463e-05
        Shapiro-Wilk normality test
data:  sa
W = 0.87765, p-value = 0.3287
```

Результаты тестирования указывают на значимые отличия от нормального распределения частот в первой выборке (**cg**): `p-value = 2.463e-05` < 0.05. Формально в этом случае следует использовать критерий Вилкоксона – Манна – Уитни (**wilcox.test()**).

```
wilcox.test(cgg,sa)
Wilcoxon rank sum test with continuity correction
data:  cg and sa
W = 180, p-value = 0.001752
alternative hypothesis: true location shift is not equal to 0
```

Выборки значимо отличаются (`p-value = 0.001752` < 0.05) – полевки (`cg`) крупнее бурозубок (`sa`).

Вместе с тем на диаграмме можно видеть четыре «выброса». Для нормализации распределения эти значения `([1] 8.7, [44] 35.8, [45] 39.5` и `[46] 55.8` упорядоченного вектора `cg`) можно исключить и повторить процедуру тестирования.

```
(cgg<-cg[2:43])
shapiro.test(cgg)
        Shapiro-Wilk normality test
data:  cgg
W = 0.97962, p-value = 0.6468
```

Теперь отличия между эмпирическим и нормальным распределениями не значимы в обоих случаях. Это открывает возможность использования параметрических критериев. Для сравнения средних арифметических используют два варианта критерия Стьюдента (**t.test()**): классический – для случая, когда дисперсии выборок однородны (не отличаются статистически, **var.equal=TRUE**), и критерий Стьюдента с поправкой Велша – для неоднородных дисперсий (по умолчанию **var.equal=FALSE**). Однородность дисперсий устанавливают по критерию Фишера (**var.test()**).

```
var.test(cgg,sa)
        F test to compare two variances
data:  cgg and sa
F = 18.56, num df = 41, denom df = 3, p-value = 0.03342
alternative hypothesis:true ratio of variances is not equal to1
95 percent confidence interval:
  1.322579 64.111194
sample estimates:
ratio of variances
          18.56011
```

```
t.test(cgg,sa, var.equal=TRUE)
        Two Sample t-test
data:  cgg and sa
t = 5.1328, df = 44, p-value = 6.211e-06
alternative hypothesis:true difference in means is not equal to0
95 percent confidence interval:
  6.751769 15.481564
sample estimates:
mean of x mean of y
 22.11667  11.00000
```

```
t.test(cgg,sa)
        Welch Two Sample t-test
data:  cgg and sa
t = 13.454, df = 18.703, p-value = 4.582e-11
alternative hypothesis:true difference in means is not equal to0
95 percent confidence interval:
  9.385456 12.847877
sample estimates:
mean of x mean of y
 22.11667  11.00000
```

В примере результаты обоих вариантов критерия Стьюдента принципиально не отличаются (`p-value = 6.211e-06` < 0.05, `4.582e-11` < 0.05) – полевки (`cg`) крупнее бурозубок (`sa`). Но поскольку дисперсии неоднородны (`p-value = 0.03342` < 0.05), то следует опираться на значения критерия Велша.

Сравнения двух выборок методом «бутстреп»

Один из вариантов ресамплинга (re- повтор, sample – проба) позволяет сравнивать выборки безотносительно от типа распределения (Шитиков, Розенберг, 2013). Нулевая гипотеза состоит в том, что различие между выбранными параметрами выбо-

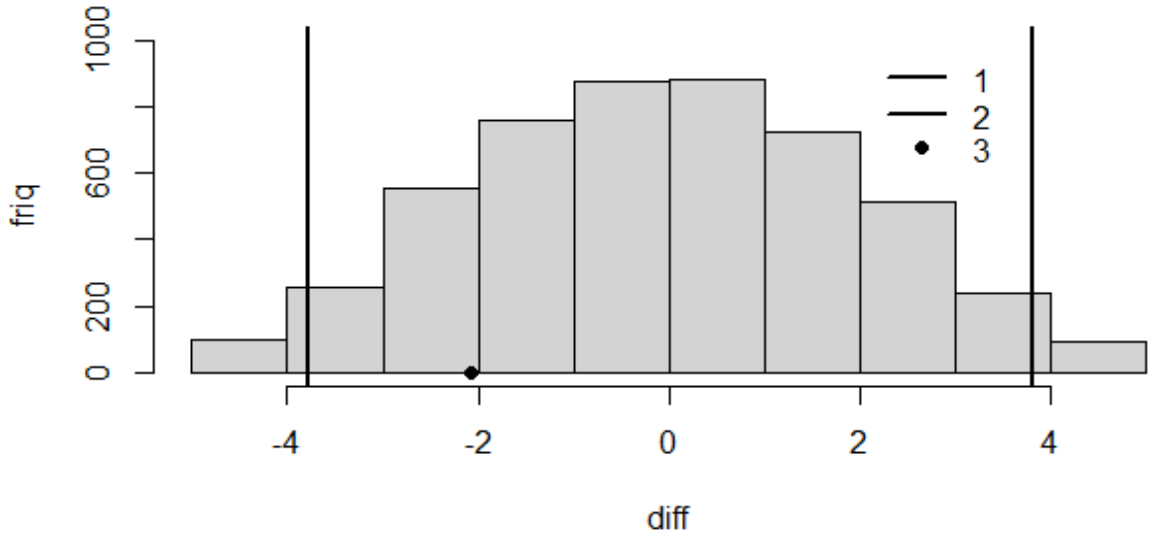
рок (в примере использована простая разница между двумя выборочными средними, **dm1**) носит случайный характер, т. е. обе выборки взяты из одной генеральной совокупности. Для проверки гипотезы генерируется распределение отличий **dm2** для 5000 пар средних, рассчитанных для выборок, взятых случайным образом из перемешанных вариант двух исходных выборок. Если первичное значение критерия **dm1** находится в 95-процентной доверительной зоне распределения **dm2**, значит, различия между выборочными средними не значимы, если за порогом, то различия считаются значимыми. В среде R задачи такого рода решаются в специальных пакетах (например, boot), но ее можно запрограммировать самостоятельно, для уяснения сути вопроса. Порядок действий следующий:

1) читаем файл с данными (формат *.csv);
2) вычисляем оценку различия между двумя исходными выборками **dm1**;
3) выполняем ресамплинг выборок, рассчитываем вторичные разности **dm2**;
4) строим гистограмму распределения переменной **dm2**;
5) определяем процентили распределения **dm2**;
6) сравниваем значение **dm1** с доверительными границами для **dm2**;
7) делаем статистический и биологический вывод по результатам сравнения.

В примере анализировалась достоверность различия между выборками массы тела рыжих полевок (**clg**) в возрасте 2 и 9 месяцев. Отличия оказались несущественными.

```
#-----чтение файла и отбор полей для сравнения-----
mm <- read.csv("mamm.csv")
cs2<-mm$spic=='cgl'&mm$ag=='2'
cs9<-mm$spic=='cgl'&mm$ag=='9'
n2 <- length(w2 <- na.omit(mm[cs2,10]))
n9 <- length(w9 <- na.omit(mm[cs9,10]))
w<-c(w2,w9)

#---разность эмпирических средних арифметических---
(dm1 <- mean(w2)-mean(w9))
[1] -2.075

#----------вторичные разности (бутстреп)-----------
dm2<-c(5000)
for (i in 0:5000)
{dm2[i]<-(mean(sample(w,size=n2))-mean(sample(w,size=n9)))}

#---------гистограмма распределения dm2-----------
hist(dm2, ylab='friq',xlab='diff',main='',ylim=c(0,1000))

#----------процентили распределения dm2-----------
(q95<-quantile(dm2, prob=c(0.025,0.975)))
 2.5%  97.5%
-3.825  3.925

abline(v=q95,lwd=2)
points (dm1,0,pch=19)
legend(2,1000,,lty=c(1,1),lwd=c(2,2),legend=c('1','2'),box.lty=0)
legend(2.35,790,legend=c('···3'),box.lty=0,pch=c(19))
```

```
#------------сравнение dm1 и dm2---------------
if(dm1<q95[1]|dm1>q95[2]) 'significant' else 'not significant'
[1] " not significant"
```



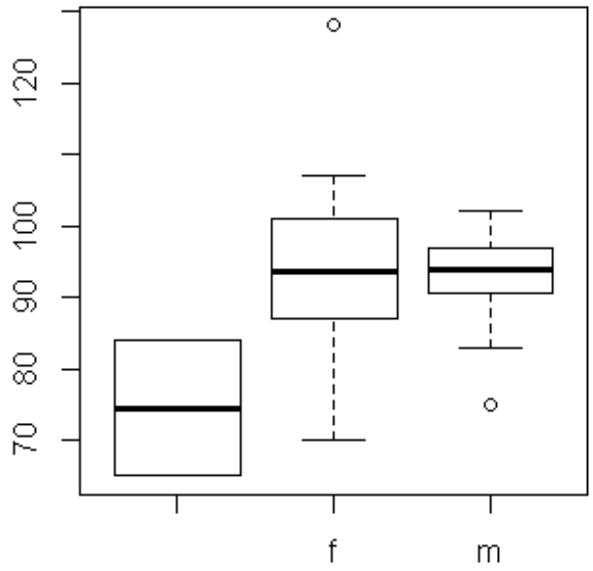
Дисперсионный анализ

Дисперсионный анализ служит для сравнения нескольких выборок, точнее говоря, для сравнения нескольких выборочных средних арифметических. Каждая из исходных выборок получена в разных условиях или для объектов разного статуса. Общая задача состоит в том, чтобы охарактеризовать силу и достоверность влияния этих условий на изучаемый признак, т. е. доказать значимость различий нескольких средних арифметических. Мы рассмотрим зависимость длины тела (**lt**) у рыжих полевок (**cgl**) от их пола (**sex**) и зрелости (**fer**).

Подготовка массива исходных данных для однофакторного дисперсионного анализа состоит в том, чтобы организовать два вектора, для двухфакторного – три вектора (**cgl[,c(11,5,6)]**). В первом векторе записаны значения изучаемого признака (**lt**, мм – 11-й столбец массива **mm**), в других – идентификаторы класса объекта (градации факторов): пол (**sex**) с двумя градациями (**'m'** и **'f'** – 5-й столбец) и зрелость (**fer**) с тремя градациями (**'juv'**, **'sad'** и **'ad'** – 6-й столбец).

```
mm<-read.csv("mamm.csv")
cgl<-mm[mm$spic=='cgl',]
head(cgl[,c(11,5,6)])
  lt sex fer
1 95   m  ad
3 91   m sad
4 99   m sad
5 86   m  ad
6 97   m  ad
7 96   f  ad
```

```
boxplot(lt~sex,cgl)
```



Дисперсионный анализ выполняет функция **aov()**. В однофакторном анализе первый аргумент (**formula**) с помощью знака тильда (**~**) выражает искомую зависимость количественного признака от градации фактора (**lt~se**), т. е. зависимость значений выборок от «дозы» изучаемого фактора. Второй аргумент указывает источник данных (массив **cgl**). Функция **aov()** выводит краткий отчет, полную таблицу дает функция **summary()**.

```
aov(lt~sex,cgl)
Call:
   aov(formula = lt ~ sex, data = cgl)
Terms:
                     sex Residuals
Sum of Squares   679.265  4181.362
Deg. of Freedom        2        48

Residual standard error: 9.333365
Estimated effects may be unbalanced
2 observations deleted due to missingness
```

```
summary(aov(lt~sex,cgl))
            Df Sum Sq Mean Sq F value Pr(>F)
se           2    679   339.6   3.899  0.027 *
Residuals   48   4181    87.1
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
2 observations deleted due to missingness
```

В отчете можно найти важные результаты: стандартную ошибку остатков (`Residual standard error`) – квадратный корень из отношения суммы квадратов к числу степеней свободы для случайной дисперсии, и оценку значимости влияния фактора на результативный признак, т. е. значимо ли отличаются факториальная дисперсия от остаточной. В примере уровень значимости `Pr(>F) = 0.027` < 0.05 (`*`), что как буд-

то бы указывает на наличие полового диморфизма. Вместе с тем перекрывание диапазонов изменчивости у самцов и самок на диаграмме противоречит этому предварительному заключению. Результаты попарного сравнения средних по критерию Тьюки (**TukeyHSD()**) показывают иное.

```
TukeyHSD(aov(lt~sex,cgl))
  Tukey multiple comparisons of means
    95% family-wise confidence level
Fit: aov(formula = lt ~ sex, data = cgl)
$sex
          diff       lwr       upr     p adj
f-  18.9545455  2.283555 35.625536 0.0224203
m-  18.6481481  2.106292 35.190005 0.0237789
m-f -0.3063973 -6.789560  6.176766 0.9928245
```

Значимые отличия полов отсутствуют (m-f: p = 0.9928245 > 0.05). Общая дифференциация выборок связана здесь с отличиями самцов и самок от особей с неопределенным полом (f- и m-: p < 0.05), представленных, скорее всего, молодыми особями, т. е. анализ фактически выявляет возрастные отличия.

Применение функции **aov()** ограничено двумя требованиями – нормальности распределения частот и однородности дисперсий в выборках. Общий порядок реализации расчётов такой же, как при сравнении двух выборок. При несоответствии первому требованию используют непараметрический дисперсионный анализ с критерием Краскела – Уоллеса (**kruskal.test()**), во втором случае, если первое требование соблюдено, – критерий Фишера по Велшу (**oneway.test()**). Соответствие нормальному закону можно оценить с помощью критерия Шапиро – Уилкса (**shapiro.test()**), сравненить дисперсии – с помощью критерия Бартлетта (**bartlett.test()**).

```
tapply(cgl$lt, cgl$sex, shapiro.test)
Error in FUN(X[[i]], ...):sample size must be between 3 and 5000
```

```
bartlett.test(lt~sex,cgl)
        Bartlett test of homogeneity of variances
data:  lt by sex
Bartlett's K-squared = 7.4255, df = 2, p-value = 0.02441
```

Проверка на нормальность в рассматриваемом примере не дала результата, поскольку особей с неустановленным полом всего две, а нужно не менее трех. Дисперсии неоднородны (p-value = 0.02441 < 0.05).

```
kruskal.test(lt~sex,cgl)
        Kruskal-Wallis rank sum test
data:  lt by sex
Kruskal-Wallis chi-squared = 4.3244, df = 2, p-value = 0.1151
```

```
oneway.test(lt~sex,cgl)
        One-way analysis of means (not assuming equal variances)
data:  lt and sex
F = 1.5332, num df = 2.0000, denom df = 2.6511, p-value = 0.361
```

Критерии Краскела – Уоллеса и Велша, указывают на отсутствие половой дифференциации особей (`p-value = 0.1151` > 0.05, `0.361` > 0.05), но в данном случае лучше исключить из массива особей с неустановленным полом и выполнить сравнение двух выборок по алгоритму, который рассмотрен на стр. 83–85.

Многофакторный дисперсионный анализ оформляется почти так же, как и однофакторный, только в формулу зависимости количественного признака от класса группы вводятся дополнительные члены, в примере – зрелость зверьков (**`fer`**). Знак **`+`** дополняет дисперсионную таблицу еще одним членом (**`lt~sex+fer`**), знак * требует оценить, кроме прямого слияния факторов, еще и их взаимодействие (**`lt~sex+fer+sex*fer`**). Более краткая форма – (**`lt~sex*fer`**).

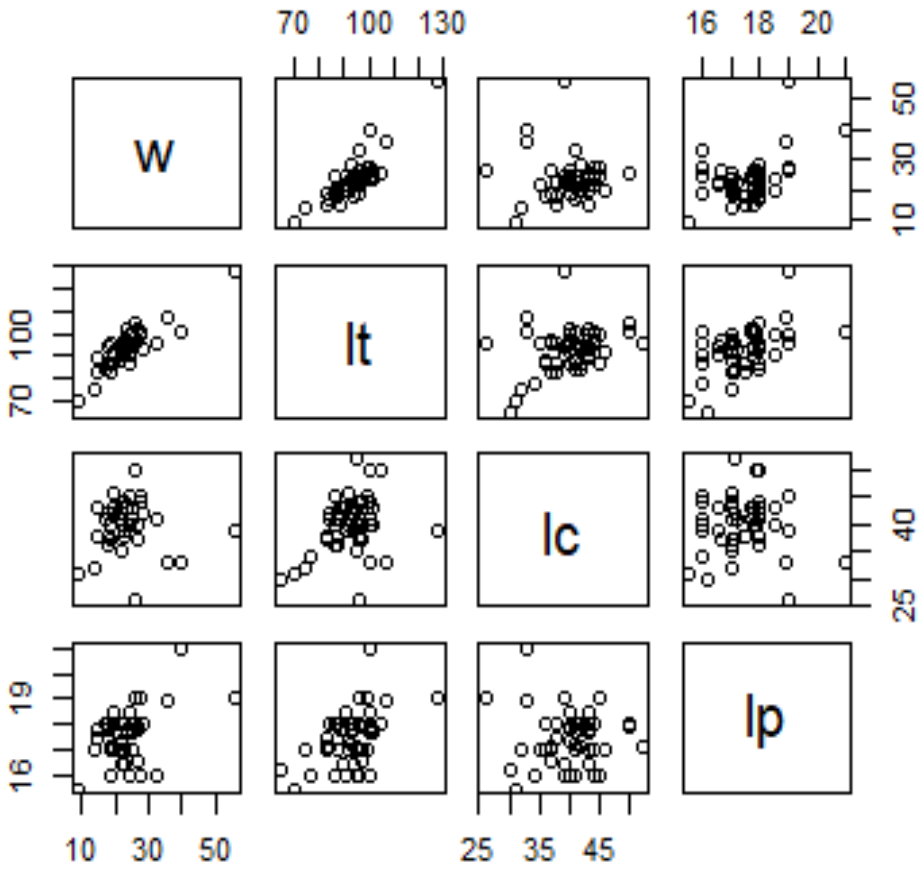
```
summary(aov(lt~sex+fer,cgl))
            Df Sum Sq Mean Sq F value  Pr(>F)
sex          2  679.3   339.6   5.502 0.00728 **
fer          3 1403.8   467.9   7.581 0.00033 ***
Residuals   45 2777.6    61.7
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
2 observations deleted due to missingness
summary(aov(lt~sex+fer+sex*fer,cgl))
            Df Sum Sq Mean Sq F value   Pr(>F)
sex          2  679.3   339.6   5.250 0.009224 **
fer          3 1403.8   467.9   7.233 0.000509 ***
sex:fer      3   60.4    20.1   0.311 0.817078
Residuals   42 2717.1    64.7
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
2 observations deleted due to missingness
summary(aov(lt~sex*fer,cgl))
            Df Sum Sq Mean Sq F value   Pr(>F)
sex          2  679.3   339.6   5.250 0.009224 **
fer          3 1403.8   467.9   7.233 0.000509 ***
sex:fer      3   60.4    20.1   0.311 0.817078
Residuals   42 2717.1    64.7
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
2 observations deleted due to missingness
```

Важно отметить, что функции **`aov()`** не нужно указывать на возможные пробелы `NA` в данных – она их автоматически исключает и сообщает в комментариях (`2 observations deleted due to missingness`). Результаты показывают, что размер полевок зависит, прежде всего, от зрелости зверьков (`fer: Pr(>F) = 0.000509` < 0.001) и во вторую очередь – от их пола (`sex: Pr(>F) = 009224` < 0.01), совместное влияние этих факторов (`sex:fer`) не значимо (`Pr(>F) = 0.817078` > 0.05). Остатки – внутригрупповая (случайная) дисперсия (`Residuals` в `Sum Sq`), хоть уменьшились по сравнению с однофакторным комплексом, но составляют более 50 % общей дисперсии признака.

Корреляция

Корреляционная зависимость вычисляется с помощью функции **cor()**, которая возвращает коэффициент корреляции Пирсона. Она может рассчитываться одновременно относительно нескольких переменных. Большую помощь в понимании источника корреляций дает диаграмма зависимости изучаемых признаков **plot()**. Для ликвидации пробелов используем функцию **na.omit()**, а для сокращения числа знаков после запятой – функцию округления (**round()**).

```
mm<-read.csv("mamm.csv")
cgs<-mm[mm$spic=='cgl',10:13]
round(cor(na.omit(cgs)),2)
      w   lt    lc    lp
w  1.00 0.86  0.00  0.44
lt 0.86 1.00  0.21  0.44
lc 0.00 0.21  1.00 -0.14
lp 0.44 0.44 -0.14  1.00
plot(cgs)
```



Значимость отличия коэффициента корреляции от нуля проверяется с помощью функции **cor.test()**, но сначала нужно удалить пробелы из массивов (условие **na.rm=TRUE** здесь не работает); при этом нужно следить за тем, чтобы векторы были одинаковой длины.

```
cg<-na.omit(mm[mm$spic=='cgl',10:11])
cgw<-cg[,1]
cgl<-cg[,2]
cor(cgw,cgl)
[1] 0.8594417
```

```
cor.test(cgw,cgl)
    Pearson's product-moment correlation
data:  cgw and cgl
t = 10.895, df = 42, p-value = 8.168e-14
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.7552694 0.9212611
sample estimates:
      cor
0.8594417
```

Зависимость между массой и длиной тела у рыжих полевок оказалась значимой (`p-value = 8.168e-14` < 0.05), коэффициент корреляции (`cor) 0.8594417` указывает на сильную положительную связь.

Использование коэффициента Пирсона предполагает двумерное нормальное распределение признаков и линейную связь между ними, если такой уверенности нет, то следует использовать ранговые коэффициенты – Спирмена или Кендалла. В этом случае в функцию **`cor()`** вводят аргумент (**`method = "spearman"`** или **`method = "kendall"`**).

```
cor.test(cgw,cgl, method = "spearman")
           Spearman's rank correlation rho
data:  cgw and cgl
S = 3101.3, p-value = 3.854e-10
alternative hypothesis: true rho is not equal to 0
sample estimates:
      rho
0.7814418
```

Прямолинейная регрессия

Рассчитаем типичный вариант зависимости массы тела (**`w`**) полевки от ее размеров (**`lt`**) в форме уравнения линейной регрессионной **`w = a1+a2*lt`**. Коэффициенты уравнения посредством метода наименьших квадратов рассчитывает функция **`lm()`**. Для наглядности отдельным векторам присваиваем отдельные имена: зависимой переменной – **`w`**, независимой – **`lt`**.

```
mm<-read.csv("mamm.csv")
w<-mm[mm$spic=='cgl',10]
lt<-mm[mm$spic=='cgl',11]
```

Вызываем функцию расчета линейной регрессии **`lm()`** и в аргументах записываем формулу *связи* между переменными (**`w~lt`**), результат расчётов сохраняем в массиве **`res`**. Получено уравнение `w = -43.0061 + 0.7085*lt`.

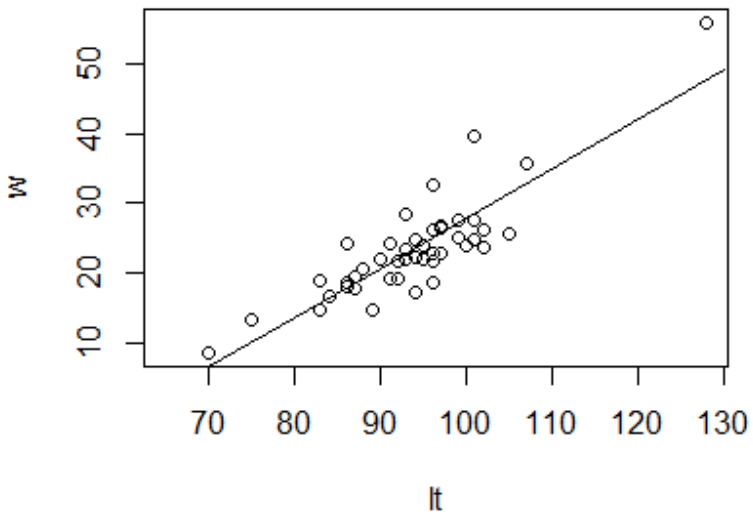
```
(res<-lm(w~lt))
Call:
lm(formula = w ~ lt)
Coefficients:
(Intercept)           lt
   -43.0061       0.7085
```

Массив результата имеет сложную структуру, с помощью функции **str()** из него можно извлечь много полезной информации.

```
str(res)
List of 13
 $ coefficients : Named num [1:2] -43.006 0.709
  ..- attr(*, "names")= chr [1:2] "(Intercept)" "lt"
 $ residuals    : Named num [1:44] -0.306 2.928 -2.14 6.371 ...
 $ fitted.values: Named num [1:44] 24.3 21.5 27.1 17.9 25.7 ...
  ..- attr(*, "names")= chr [1:44] "1" "2" "3" "4" ...
 $ model        :'data.frame':  44 obs. of  2 variables:
  ..$ w : num [1:44] 24 24.4 25 24.3 23 32.7 18.7 27.6 ...
  ..$ lt: num [1:44] 95 91 99 86 97 96 86 101 102 96 ...
```

Блок `$coefficients` содержит собственно искомые коэффициенты, в блок `$residuals` помещены отклонения реальных значений массы от прогноза по модели, блок `$fitted.values` включает рассчитанные по модели значения массы тела, в блок `$model` включены исходные данные в формате **data.frame**. Если к имени массива с результатами через знак доллара добавить имя блока, с ним можно дальше работать. В частности, часто бывает полезным использовать коэффициенты в прямых расчетах теоретических значений.

```
(a<-res$coefficients)
(Intercept)           lt
-43.0060921   0.7085457
x<-c(70,90,100,130)
round((a[1]+a[2]*x),1)
[1]  6.6 20.8 27.8 49.1
plot(lt,w)
points(x,(a[1]+a[2]*x), type='l')
```



Рассчитывать теоретические значения для исходных данных часто удобно с помощью специальной функции **predict()**, эти же значения приведены в блоке `$fitted.values`.

```
predict(res)[2]
       2
21.47157
```

```
round((res$fitted[c(1:3,10:12)]),2)
  1      2      3      10    11     12
24.31  21.47  27.14  25.01  24.31  20.05
```

Если же нужно рассчитать теоретические значения для иных значений независимых переменных, сначала нужно подготовить массив этих данных в формате **data.frame**, причем обязательно указать то же имя для независимой переменной, что было в формуле регрессии (у нас **lt**), после чего подставить его в функцию.

```
neww <- data.frame(lt = c(100,110,120))
predict(lm(w~lt),newdata = neww)
    1        2        3
27.84848 34.93393 42.01939
```

Результаты статистического анализа дают функции **aov()** и **summary()**.

```
aov(res)
Call:
   aov(formula = sres)
Terms:
                      lt Residuals
Sum of Squares  1771.7684  626.9214
Deg. of Freedom         1        42

Residual standard error: 3.863509
Estimated effects may be unbalanced
9 observations deleted due to missingness
```

```
(sres<-summary(res))
Call:
lm(formula = w ~ lt)
Residuals:
    Min      1Q  Median      3Q     Max
-6.3143 -2.4100 -0.0673  1.2663 10.9430
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -43.00609    6.12772  -7.018 1.37e-08 ***
lt            0.70855    0.06503  10.895 8.17e-14 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.864 on 42 degrees of freedom
  (9 observations deleted due to missingness)
Multiple R-squared:  0.7386,    Adjusted R-squared:  0.7324
F-statistic: 118.7 on 1 and 42 DF,  p-value: 8.168e-14
```

Все необходимые для выводов сведения можно извлечь из массива результатов анализа: кэффициенты регрессии (`Coefficients`) значимо отличаются от нуля (`Pr(>|t|) = 1.37e-08` < 0.001, `8.17e-14` < 0.001), критерий Фишера (`F-statistic`) показывает, что регрессионная дисперсия значимо больше остаточной (`p-value = 8.168e-14` < 0.001), коэффициент детерминации (`R-squared`) составил `0.7386`, остаточное стандартное отклонение (`Residual standard error`) равно `3.864`.

Множественная регрессия

Линейную зависимость одного признака от серии независимых признаков можно рассчитать с помощью той же функции **lm()**. В этом случае формула зависимости дополняется новыми переменными, кроме того, возможно добавить эффекты их взаимодействия. Рассмотрим связь между массой тела (**w**), длиной тела (**lt**) и длиной хвоста (**lc**) у рыжей полевки (**cgl**).

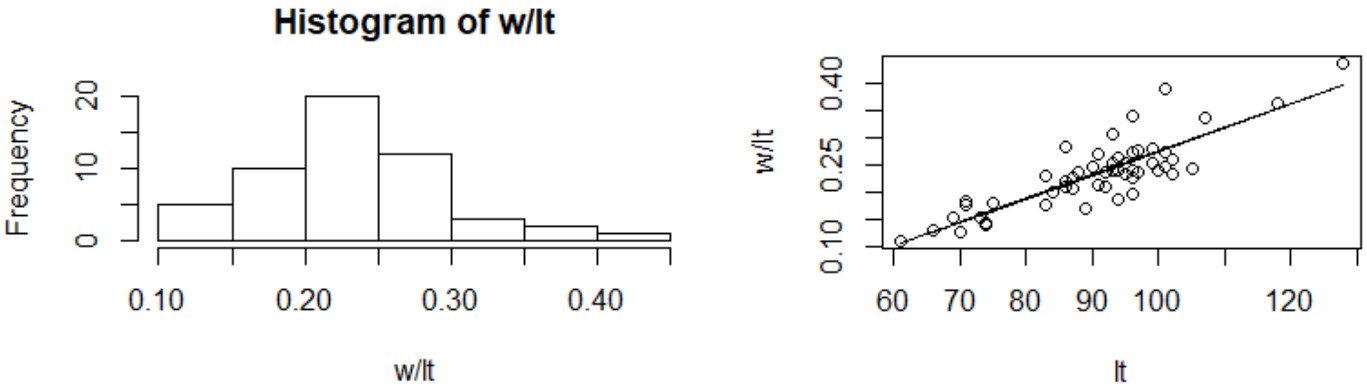
```
summary(lm(w~lt+lc))
Call:
lm(formula = w ~ lt + lc)

Residuals:
    Min      1Q  Median      3Q     Max
-7.2831 -2.3048 -0.3201  1.7542  8.6012
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -33.67373    7.04479  -4.780 2.27e-05 ***
lt            0.73903    0.06313  11.706 1.18e-14 ***
lc           -0.30514    0.12972  -2.352   0.0235 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.67 on 41 degrees of freedom
  (9 observations deleted due to missingness)
Multiple R-squared:  0.7697,    Adjusted R-squared:  0.7585
F-statistic: 68.52 on 2 and 41 DF, p-value: 8.438e-14
```

Как можно видеть, новый признак (lc) имеет значимый коэффициент корреляции, однако его добавление очень незначительно улучшило общую корреляцию модели с реальностью (коэффициент детерминации увеличился всего на три сотых – с 0.7386 до 0.7697). В то же время интерпретация коэффициентов резко ухудшилась. Получилось, что масса тела имеет отрицательную зависимость от длины хвоста (чем меньше хвост, тем больше масса тела), что с биологической точки зрения абсурдно. Причина состоит в том, что длина тела и длина хвоста тесно взаимозависимы (см. стр. 91–92). В уравнении множественной регрессии эта общая корреляция была учтена в коэффициенте регрессии для длины тела, на долю коэффициент регрессии для длины хвоста остались аллометрические зависимости (диспропорции среди выборки животных). Эту особенность необходимо учитывать и по возможности не включать в множественный регрессионный анализ в качестве независимых коррелирующие признаки.

Криволинейная регрессия

В среде R есть возможность вычислять коэффициенты криволинейной регрессии методами оптимизации, т. е. с помощью прямой подгонки, т. е. поиском лучших коэффициентов, перебирая возможные значения. Для этого достаточно удобна функция **nls()**. Биологам часто интересны соотношения в пропорциях особей, например упи-

танность животных, т. е. отношение массы (`w`) к размерам тела (`lt`). Рассмотрим зависимость упитанности (`w/lt`) от размеров тела для всех пяти видов нашей выборки. Одна из проблем при изучении этого феномена состоит в том, что более крупные животные непропорционально много весят. Результатом является правосторонняя асимметрия индекса и отклонение точек от линейной регрессии. В этом случае приходится пользоваться методами криволинейной регрессии.



Подготовим исходные данные, предварительно удалив пробелы из нужных векторов и рассчитав значения индекса упитанности.

```
mm<-read.csv("mamm.csv")
dat<-na.omit(mm[,c(10,11)])
w<-dat[,1]
lt<-dat[,2]
wlt<-w/lt
```

Далее выполним регрессионный анализ для линейного, параболического и степенного уравнения. Функция `nls()` требует ввода точной формулы зависимости между переменными и определения начальных значений коэффициентов регрессии в формате списка (`list()`). Функция позволяет оценивать параметры для самых разных уравнений (в примере – линейная, параболическая, степенная), однако она чувствительна к стартовым значениям, особенно для уравнений, содержащих степени и логарифмы. Необходимо задавать правдоподобные значения, которые программа уточнит в процессе настройки регрессионной модели.

```
regl<-nls(wlt~a[1]+lt*a[2],start=list(a=c(1,1)))
regp<-nls(wlt~a[1]+a[2]*lt+a[3]*lt^2,start=list(a=c(1,1,1)))
regm<-nls(wlt~a[1]*lt^a[2],start=list(a=c(0.2,1.5)))
```

Дисперсионный анализ показывает, что коэффициенты линейной модели значимы, параболической – не значимы, степенной – значимы.

```
summary(regl)
Formula: wlt ~ a[1] + lt * a[2]
Parameters:
     Estimate Std. Error t value Pr(>|t|)
a1 -0.1596366  0.0350595  -4.553 3.31e-05 ***
a2  0.0043369  0.0003832  11.319 1.60e-15 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 0.03494 on 51 degrees of freedom
Number of iterations to convergence: 1
Achieved convergence tolerance: 3.015e-10
```

```
summary(regp)
Formula: wlt ~ a[1] + a[2] * lt + a[3] * lt^2
Parameters:
    Estimate Std. Error t value Pr(>|t|)
a1 5.910e-03  1.516e-01   0.039    0.969
a2 5.993e-04  3.352e-03   0.179    0.859
a3 2.069e-05  1.843e-05   1.122    0.267
Residual standard error: 0.03485 on 50 degrees of freedom
Number of iterations to convergence: 2
Achieved convergence tolerance: 2.537e-07
```

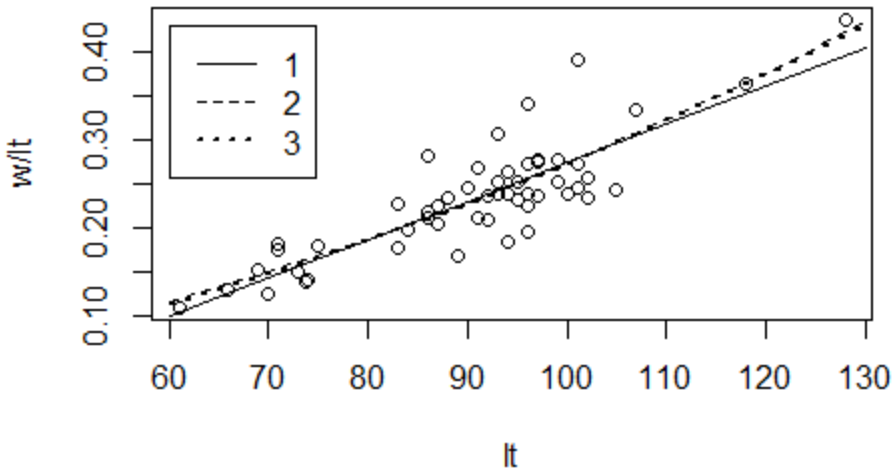
```
summary(regm)
Formula: wlt ~ a[1] * lt^a[2]
Parameters:
    Estimate Std. Error t value Pr(>|t|)
a1 9.866e-05  6.883e-05   1.433    0.158
a2 1.721e+00  1.530e-01  11.253 1.98e-15 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
Residual standard error: 0.03453 on 51 degrees of freedom
Number of iterations to convergence: 15
Achieved convergence tolerance: 4.698e-06
```

Для создания диаграммы предварительно зададим значения независимого признака **lt**, для которых будут рассчитываться теоретические значения индексов.

```
new<-data.frame(lt=seq(60,130,10))
new[,1]
[1]  60  70  80  90 100 110 120 130
```

```
plot(lt,w/lt)
points(new[,1],predict(regl,new),type='l')
points(new[,1],predict(regp,new),type='l',lty=2,lwd=1)
points(new[,1],predict(regm,new),type='l',lty=3,lwd=2)
legend(60,.43,c(1,2,3),lty=c(1,2,3),lwd=c(1,1,2))
```

На диаграмме видно, что степенная модель подошла к крайним точкам ближе, чем линейная, значит, ее можно взять в качестве более подходящей.



Метод главных компонент

Компонентный анализ способен представить структуру отношений многомерных данных, вычленяя группы (плеяды) зависимых признаков и группы (кластеры) сходных объектов. На основе исходных признаков он рассчитывает линейные индексы (главные компоненты), коэффициенты в которых показывают, насколько сильно исходные признаки коррелируют друг с другом. Новые показатели выражают некие общие причины, из-за которых группы признаков изменяются согласованно, а группы объектов оказываются сходными. В числе таких общих факторов может быть возраст, пол, состояние здоровья, генетическое родство, воздействие абиотических факторов, в том числе антропогенных. Мы рассмотрим всех особей нашей выборки с целью оценить их отличие друг от друга по морфологическим признакам.

Вначале загрузим только видовые названия и промеры, исключив (**`na.omit()`**) пробелы в данных. Затем центрируем и нормируем данные (**`scale()`**), чтобы анализ работал с корреляционной матрицей. Функция **`eigen()`** использует ковариационную матрицу (**`var`**) и рассчитывает дисперсии компонент (собственные значения, блок `$values`), а также и факторные нагрузки (собственные векторы, блок `$vectors`). Функция **`princomp()`** выполняет компонентный анализ и позволяет рассчитать значения главных компонент (**`predict()`**). В принципе то же самое получим, если матрицу нормированных значений умножим на матрицу нагрузок: **`m[1:nrow(m),]%*%m.e$vectors`**.

```
head(mm<-na.omit(read.csv("mamm.csv")[,c(2,10:13)]),2)
  spic    w lt lc lp
1  cgl 24.0 95 40 16
2  sar 12.4 71 36 12
```

```
head(m <- scale(mm[,2:5]),3)
           w          lt         lc         lp
1  0.2450194  0.34501483 -0.1859226 -0.4400016
2 -1.0705958 -1.55316363 -0.5116714 -2.3752785
3  0.2903855  0.02865175 -0.2673598 -0.4400016
```

```
(m.e <- eigen(var(m)))
eigen() decomposition
$values
[1] 2.52743067 1.03229184 0.35773909 0.08253841
$vectors
           [,1]         [,2]       [,3]        [,4]
[1,] -0.5887610 -0.008054622 -0.5039944  0.63189025
[2,] -0.6070578 -0.026249691 -0.2397422 -0.75717597
[3,]  0.2194409  0.894602935 -0.3794699 -0.08679801
[4,] -0.4865050  0.446017563  0.7379132  0.14094457
```

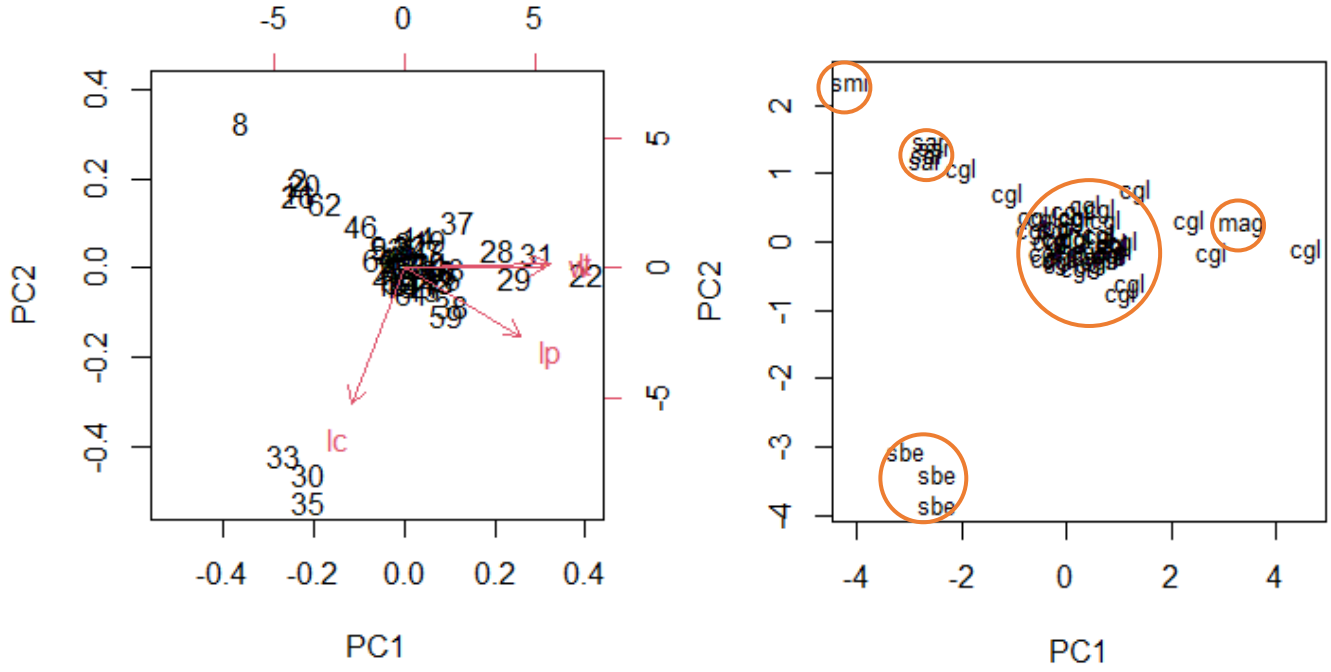
```
(m.a <- princomp(m))
Call:
princomp(x = m)
Standard deviations:
   Comp.1    Comp.2    Comp.3    Comp.4
1.5747201 1.0063869 0.5924435 0.2845717
 4  variables and  53 observations.
```

```
head(m.pc <- predict(m.a),3)
       Comp.1       Comp.2    Comp.3     Comp.4
1  0.18043785  0.373605454 0.4603340 -0.1522897
2 -2.61648833  1.467765331 0.6466522  0.2091485
3  0.03296758  0.438520369 0.3764497  0.1229878
```

```
head(m[1:nrow(m),]%*%m.e$vectors,2)
          [,1]          [,2]       [,3]        [,4]
1 -0.18043785 -0.373605454 -0.4603340 -0.1522897
2  2.61648833 -1.467765331 -0.6466522  0.2091485
```

```
biplot(m.a, xlab="PC1", ylab="PC2")
```

```
plot(m.pc[,1:2], type="n", xlab="PC1", ylab="PC2")
text(m.pc[,1:2], labels=mm[,1],cex=.8)
```



Все виды расположились в осях двух первых главных компонент вполне обособленно (кружки нарисованы нами в редакторе растров), за исключением рыжей

полевки. Эти зверьки, во-первых, были разной зрелости, во-вторых, по крайней мере у двух зверьков вид был определен неправильно.

Динамические имитационные модели

Коренное отличие таких моделей от регрессий состоит в том, что динамическая модель призвана описывать некие процессы, на каждом временно́м шаге она рассчитывает каждое новое свое значение, основываясь на теоретически определенном приращении, полученном на предыдущем шаге. Иными словами, модель задается двумя формулами:

$dy_i = f(a_j, x_i, my_i)$,
$my_{(i+1)} = my_i + dy_i$,
где dy_i – приращение значения модели на данном i-м шаге,
a_j – коэффициенты пропорциональности,
x_i – значения независимых переменных (внешней среды),
my_i – значение модели на текущем i-м шаге,
$my_{(i+1)}$ – значение модели на следующем шаге.

Первое уравнение рассчитывает приращение модели, второе – новое значение модели.

Зачастую представляет интерес автономный процесс, когда текущее приращение модели зависит от текущего модельного значения (здесь – линейно):

$$dy_i = a_1 + a_2 * my_i.$$

Свою лепту могут вносить и некие внешние факторы, имеющие разную выраженность на разных шагах модели (в разные моменты времени):

$$dy_i = a_1 + a_2 * my_i + a_3 * x_i.$$

Весь смысл имитационного моделирования состоит в том, чтобы *настроить модель*, т. е. подобрать такие коэффициенты a, чтобы модель хорошо описывала фактические данные. Это требование, выраженное количественно, звучит как минимизация функции невязки, т. е. сведение к нулю суммы квадратов отклонения модельных значений от эмпирических: $\Sigma (y - my)^2 \rightarrow 0$. В среде R эту операцию можно выполнить разными способами, мы воспользуется функцией `nlm()`.

Компоненты имитационной системы

При создании имитационной динамической модели приходится программировать решение несколько задач, образующих вместе с программированием самой модели некую программную среду – имитационную систему. В комплекс решаемых задач входят следующие:

1) организовать массив исходных данных,

2) организовать последовательности шагов динамической модели,
3) составить модельные формулы,
4) назначить правдоподобные стартовые значения параметров модели,
5) решить проблему пропуска данных и повторяемости данных для отдельных модельных шагов,
6) составить формулу функции невязки,
7) создать пользовательскую целевую функцию расчета функции невязки,
8) выполнить настройку модели,
9) рассчитать и визуализировать результаты моделирования.

Модель динамики роста массы тела рыжей полевки с возрастом

Исходные данные. Используя наши данные, рассмотрим рост рыжей полевки в зависимости от возраста. Отбираем малочисленные данные.

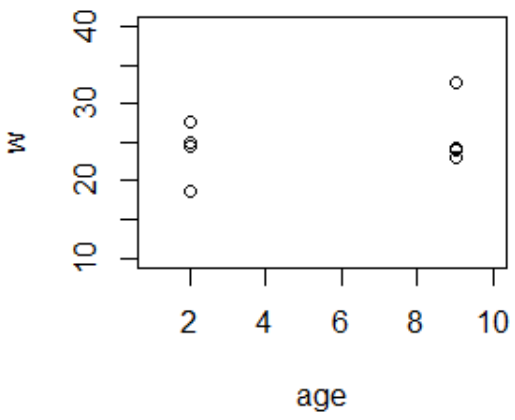
```
mm<-read.csv("mamm.csv")
head(mm,n=2)
    n spic year season sex fer age    yeseli line    w lt lc lp
1 976  cgl 2001      6   m  ad   9 200106002    2 24.0 95 40 16
2 977  sar 2001      6   f  ad   1 200106002    2 12.4 71 36 12
```

```
  (mc<-na.omit(mm[mm$spic=='cgl',c(7,11,10)]))
   age    w
1    9 24.0
3    2 24.4
4    2 25.0
5    9 24.3
6    9 23.0
7    9 32.7
9    2 18.7
10   2 27.6
```

Запишем в массив **y** все исходные данные для моделирования: `age` – возраст в месяцах, `w` – масса тела в граммах. Для наглядности отсортируем данные по возрасту.

```
y<-mc[order(mc[,1]),]; y
   age    w
3    2 24.4
4    2 25.0
9    2 18.7
10   2 27.6
1    9 24.0
5    9 24.3
6    9 23.0
7    9 32.7
```

```
plot(y,xlim=c(1,10),ylim=c(10,40))
```



Модельный массив. Будем строить модель увеличения массы тела полевки (**my**) в течение 10 месяцев с шагом 1 месяц (**i**). В данном разделе рассмотрен алгоритм, в котором модельные данные начинаются с отсчета 1 и с каждым шагом увеличиваются на 1. Если данные имеют другие стартовое значение и величину шага, их нужно привести к этой схеме. В случае нашей модели условие выполнено: рост начинается с 1 месяца и будет рассчитываться для каждого месяца. Эмпирические данные содержат значения массы только для двух отсчетов (возрастов) – для 2 и для 9. Однако модельные значения должны быть рассчитаны для всех десяти шагов (месяцев).

Создадим массив, в который будут записаны модельные значения. Он должен быть подобен массиву эмпирических данных, но содержать теоретические значения массы (**my**) для всех предусмотренных возрастных шагов (**s**) длиной в 1 месяц.

```
(my<-data.frame(s=c(1:10),my=rep(NA,1,10)))
    s my
1   1 NA
2   2 NA
3   3 NA
4   4 NA
5   5 NA
6   6 NA
7   7 NA
8   8 NA
9   9 NA
10 10 NA
```

В начале этот массив не содержит значений массы. В качестве стартового модельного значения массы необходимо принять какую-либо правдоподобную величину, ориентируясь на диаграмму эмпирических данных, например 21. Эта величина также является настраиваемым параметром и должна быть включена в список параметров **p<-c(0,.04,21)** (см. ниже). В начале работы модели это значение будет задано как стартовое **my[1,2]<-p[3]**.

Формулы модели. Масса тела **my[i+1,2]** в каждый последующий месяц (**i+1**) будет вычисляться как сумма массы тела **my[i,2]** в предыдущий месяц (**i**) и прибавки **dy** за данный месяц. Прибавка, в свою очередь, задается как линейная функция от достигнутой массы тела **my[i,2]** (с настраиваемыми коэффициентами **p**).

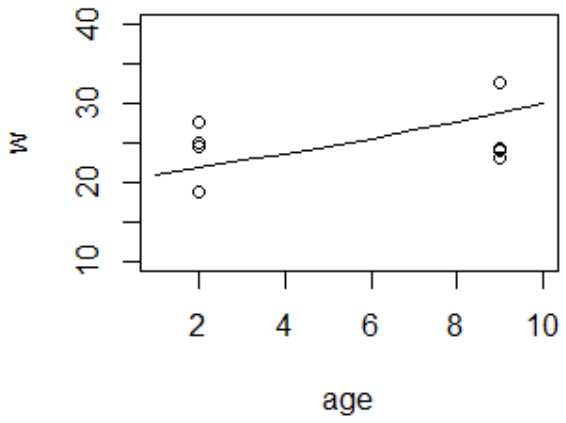
```
dy <- p[1]+p[2]*my[i,2]
my[i+1,2] <- my[i,2]+dy
```

Модельные данные будут записываться в массив **my**, длина которого равна 10. Поскольку модель должна рассчитать 9 новых значений с опорой на предыдущие, приходится использовать цикл.

```
for (i in 1:9){dy <- p[1]+p[2]*my[i,2]; my[i+1,2] <- my[i,2]+dy}
```

Стартовые значения параметров. Значения параметров **p** при последующей настройке модели изменятся, но перед настройкой следует задать биологически осмысленные, правдоподобные значения, иначе поиск может застрять в локальном минимуме функции невязки. Диаграмма может служить контролем для первичного назначения значений параметров.

```
p<-c(0,.04,21)
my[1,2]<-p[3]
for (i in 1:9){dy <- p[1]+p[2]*my[i,2]; my[i+1,2] <- my[i,2]+dy}
lines(my[,2])
```



Ручной перебор значений позволил получить примерные значения **p<- c(0,.04,21).** Полученный ход модели примерно соответствует ожиданиям – со 2-го по 9-й месяцы масса зверьков увеличивается.

Проблема пропуска и повтора данных. Невязка – это отличия между эмпирическим и модельным рядами данных. Однако эти ряды отличаются и по размеру, и по составу. Эмпирические значения в виде серии повторяющихся вариант известны только для шагов 2 (4 варианты) и 9 (4 варианты), тогда как модель рассчитывает по одному значению для одного месяца, но зато для всех шагов 1, 2, … 10. Получается, что задача состоит в том, чтобы извлечь из массива модельных значений значения **my** именно для шагов 2 и 9 и сравнивать сравнимые значения. В таблице эмпирических данных номера нужных шагов (2 и 9) указаны в первом столбце – **y[,1]**. Используя эти индексы, вызываем соответствующие модельные значения **my[y[,1],2]**, которые стоят на 2-м и 9-м местах в массиве **my**.

```
y[,1]
[1] 2 2 2 2 9 9 9 9

y[,2]
[1] 24.4 25.0 18.7 27.6 24.0 24.3 23.0 32.7
```

```
my[,2]
[1] 23.63005 23.92501 24.22034 24.51603 24.81209 25.10851
[7] 25.40531 25.70246 25.99999 26.29789

my[y[,1],2]
[1] 23.92501 23.92501 23.92501 23.92501 25.99999 25.99999
[7] 25.99999 25.99999
```

Поскольку эмпирический массив несет несколько значений для одного и того же возраста, то использование индекса `y[,1]` для массива `my` вызовет две серии одинаковых значений, соответствующих этим возрастам (4 для возраста 2 и 4 для возраста 9). Теперь сравниваемые реальный и модельный массивы обрели одинаковую длину и состав и могут быть использованы для составления функции невязки.

Формула функции невязки. Величина невязки, как суммарное отличие модели от реальности, примет вид

```
f <- sum(y[,2]-my[y[,1],2])^2.
```

Функция расчета невязки. Создадим целевую функцию с единственным аргументом – внутренней переменной, коэффициентами `p`. Функция будет использовать заранее созданный массив `my`.

```
goal<-function(p)
{
my[1,2]<-p[3]
for(i in 1:9){ dy <- p[1]+p[2]*my[i,2];my[i+1,2] <- my[i,2]+dy }
f<-sum((y[,2]-my[y[,1],2])^2)
return(f)
}
```

Важно отметить, что расчет функции невязки будет выполняться для всех эмпирических данных, даже если для одних модельных шагов нет данных, а для других имеются неодинаковые по объему выборки эмпирических значений. Проблема решается, если в формулу вызова модельного значения (`my[y[,1],2]`) ввести индекс номера модельного (`y[,1]`) шага, соответствующего текущему эмпирическому значению.

Вызовем функцию и убедимся, что она рассчитывает и возвращает значение невязки.

```
goal(p)
[1] 150.3857
```

Настройка модели. Выполняется с помощью функции `nlm()`, в списке аргументов которой должны присутствовать, во-первых, целевая функция, требующая обнуления (минимизации) (`goal()`), во-вторых, аргументы целевой функции (в примере `p`).

```
nlm(goal,p)
$minimum
[1] 102.9675
$estimate
[1]  0.265694870  0.001238492 23.630053543
$gradient
[1] -4.317658e-04  3.936591e-05  4.412549e-05
$code
[1] 2
$iterations
[1] 21
```

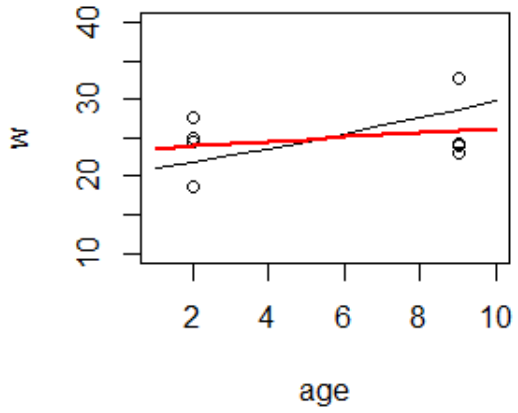
В блоке `$estimate` содержатся значения искомых коэффициентов, которые обеспечивают оптимальное описание моделью исходных данных.

Вывод результатов. Читаем параметры в массив **a** и используем для расчета модельных значений в цикле.

```
a<-nlm(goal,p)$estimate
my[1,2]<-a[3]
for(i in 1:9){dy <- a[1]+a[2]*my[i,2];my[i+1,2] <- my[i,2]+dy}
lines(my[,2],col=2)
```

Полная имитационная система будет иметь следующий код.

```
mm <-read.csv("mamm.csv")
(mc<-na.omit(mm[mm$spic=='cgl',c(7,10)]))
y<-mc[order(mc[,1]),]
plot(y,xlim=c(1,10),ylim=c(10,40))
#---------------------------------------
(my<-data.frame(s=c(1:10),my=rep(NA,1,10)))
p<-c(0,.04,21)
#-----------------------
goal<-function(p)
{my[1,2]<-p[3]
for(i in 1:9) {dy <- p[1]+p[2]*my[i,2];
my[i+1,2] <- my[i,2]+dy}
g<-sum((y[,2]-my[y[,1],2])^2)
return(g)}
#-----------------------
(a<-nlm(goal,p)$estimate)
#-----------------------
my[1,2]<-a[3]
for (i in 1:9){dy <-
a[1]+a[2]*my[i,2];my[i+1,2]<- my[i,2]+dy}
lines(my[,2],col=2,lwd=2)
```



Биологический смысл полученных коэффициентов (`0.26`, `0.00`, `23.6`) состоит в следующем. В возрасте 1 месяца полевки имеют массу 23.6 г и начинают медленный

рос со скоростью 0.26 г/мес., причем независимо от ранее достигнутой массы. Каждое из этих утверждений является ложным. Старт роста начинается примерно с 15 г в месячном возрасте, скорость имеет порядок 2 г/мес. и зависит от достигнутой массы. Причиной несоответствия интерпретаций стал очень небольшой объем данных, на основе которых построена модель. Чтобы получить гораздо более полноценное количественное описание роста, необходимо просто увеличить объем информации, относящейся к разным возрастам, и выполнить моделирование по рассмотренной схеме.

СПИСОК ФУНКЦИЙ

Назначение	Функция	Стр.
Общие		
вызвать справку по функции	**?имя_функции**	10, **21**
загрузить пакет программ	**install.packages**("имя")	
подключить пакет программ	**library**("имя")	
поиск пакета с нужной функцией ("sos")	**findFn**(string="имя")	
Управление рабочим каталогом		
задать рабочий каталог (папку)	**setwd**('диск:/каталог')	21, **30**
вывод пути до рабочей папки	**getwd()**	
вывод списка файлов в рабочей папке	**dir()**	
Чтение файла		
диалог выбора имени файла	**file.choose()**	
чтение файла в формате *.csv	**read.csv('file.csv')**	**30**, 59, 71
запись массива в файл формата *.csv	**write.csv(x,'file.csv')**	
чтение данных с клавиатуры	**scan()**	
Константы **?Constant**		
пустота (not available)	**NA**	**32**, **74-80**
символы латиницы (под номером **n**)	**LETTERS(n), letters(n)**	**37**
математические	**pi, 1, 2**…	**61**
логические	**TRUE, FALSE**	**32-34**, 39, 46
Обращение к данным **?Extract**		
назначение имени **x** массиву **a**	**x<-a**	**9**, **27**
вызов значения из **i** ячейки массива **x**, например, из третьей ячейки	**x[i], x[3]**	**9**, 12, 27, 31
задать блок (диапазон) значений	**5:12**	**12**
вызов из массива **x** столбца **a**	**x$a**	**9**, **30**, 42, 81-82
вызов значения из **j** ряда **i** столбца массива **x**, например, из **3** ряда **2** столбца	**x[j,i], x[3,2]**	**30**, 34, 56, 59
извлечь диапазон данных из массива **x**	**x[диапазон]**	27, **30**, 82
извлечь данные по условию из массива **x**	**x[условие]**	**61-65**
извлечь данные по условию	**subset(массив, условия)**	
Математические операции **?Arithmetic, ?Trig,?plotmath, ?Syntax**		
арифметические	*** / + - ^ %%**	10, 11, 96
тригонометрические	**sin(x), tg(x)**…	
алгебраические (логарифм, корень, абсолютное число …)	**log(x), sqrt(x), abs(x), exp(x)**	
округление **x** до **n** десятичных знаков	**round(x,n)**	53, 63, 78, 79, 91, 93
Логические операции **?Comparison, ?Logic**		
сравнение векторов	**> < == <= >= !=**	**62-63**
логическое И (для двух или ряда сравнений)	**&**	**64-65**
логическое ИЛИ (для двух сравнений)	**|**	**64-65**

Создание одномерных массивов (векторов) **?vector**		
список объектов в памяти	**ls()**	
удаление объекта **x** из активной памяти	**rm(x)**	
«сцепление» значений (векторов) в один вектор	**c(…,…,…,)**	**9**, **27**
стрелка «присвоение»: объект (**a**) из правой части обретает новое имя (**x**), указанное в левой части	**x<-a**	**9**, **24**, **27**
создание массива как последовательности значений от **a** и до **b** с шагом **s**	**seq(from=a,to=b,by=s)**	12, **25**
повторение **n** раз значения **x**	**rep(x,n)**	**25**
извлечение уникальных значений из массива **x**	**unique(x)**	**35**, 39
создает **n** случайных чисел в интервале 0-1; распределение равномерное	**runif(n)**	**26**
случайно выбрать **n** значений из вектора **x**	**sample(x,n)**	**26**, 79, 86
Создание двумерных массивов		
создать матрицу из вектора **x** заданной размерности (**r** рядов, **c** столбцов)	**array(x, dim=c(r,c)**	**28**, **37**, 41
объединить векторы **x1, x2,**… в таблицу	**data.frame(x1,x2,…)**	**28**, 39, 53, 101
добавить ряды **a** к массиву **x**	**append(x,a)**	
объединить ряды одинаковой длины	**rbind(x, y)**	29
объединить векторы одинаковой длины	**cbind(x,y)**	29, **57**
транспонировать массив	**t(x)**	
объединить две таблицы по общему ключу **k**	**merge(x,y,by=k)**	**66-67**, 72
Определение характеристик объекта		
характеристика структуры массива **x**	**str(x)**	12, 13, **32**, 36-37, 59
определение длины вектора **x**	**length(x)**	39, 85
число рядов и столбцов массива **x**	**nrow(x), ncol(x)**	30
вывод **n** первых и последних рядов массива **x**	**head(x,n), tail(x,n)**	**30**, 52, 65
запрос имен столбцов и строк массива **x**	**dimnames(x)**	**38**
определение и замена имен рядов и столбцов массива **x**	**names(x), colnames(x), rownames(x)**	**38**, 75
уникальные значения (без дублей)	**unique(**вектор**)**	**36**, 39
возвращает индекс ячейки искомого значения в векторе **x**	**which.max(x)** **which(...==...)**	
Изменение порядка расположения данных		
обратный порядок значений в массиве **x**	**rev(x)**	**25**, 37
сделать сортировку значений в массиве **x**	**sort(x)**	48, 79, 83
список индексов отсортированных значений массива **x**	**order(x)**	**67**, 79
сортировать по годам и месяцам	**setorder(x,y,m)**	
Определение и изменение типа данных **?is,?as**		
определение типа данных вектора **x**: целое, десятичное, текстовое, фактор, логическое?	**is.integer(x),** **is.numeric(x),** **is.character(x)** **is.factor(x),** **is.logicl(x)**	**33**
преобразовать значений вектора **x** в целое,	**as.integer(x),**	**34**, 76-77

десятичное, текстовое, фактор, логическое	**as.numeric(x), as.character(x) as.factor(x), as.logicl(x)**	
Определение и изменение класса объекта ?is,?as		
объект **x** – вектор, таблица, матрица, список?	**is.vector(x), is.data.frame(x), is.matrix(x), is.list(x)**	**36**
создать из объекта **x** вектор, таблицу, матрицу, список	**as.vector(x), as.data.frame(x), as.matrix(x), as.list(x)**	**36**, **69**
Определение и обработка пропусков ?NA		
проверить наличие в массиве **NA**	**is.na(x)**	**76**, 79
ликвидировать ячейки со значениями **NA**	**na.omit(x)**	40, 48, 63, 85
индексы ячеек со значениями **NA**	**na.action(na.omit())**	**74-75**
аргумент для игнорирования пропусков	**na.rm=TRUE**	39, 82
Обработка текстовых данных		
объединение подстрок без пробела	**paste0(a,b)**	
объединение подстрок с пробелом	**paste(a,b)**	
извлечь или заменить часть текста	**substring(a,границы), substr(a,границы)**	
длина строки	**nchar (a,b)**	
Сравнение и обработка пересечения объектов		
равны ли векторы?	**setequal(x, y)**	
значения, общие для двух векторов	**intersect(x,y)**	
индексы значений **y**, совпадающих с **x**	**match(x,y)**	
значения из **x**, которых нет в **y**	**setdiff(x,y)**	
индекс ячейки искомого значения в векторе	**which.max(), which(x==a)**	
объединение рядов с удалением дублей	**union(x, y)**	
объединить две таблицы по общему ключу **k**	**merge (x,y,by=k)**	**67**, 72
Операции (FUN) над массивами		
обобщить все записи по рядам или полям (**MARGIN=**1 или 2) с помощью функции	**apply(x,MARGIN=,FUN=…)**	40
обобщить записи одного поля массива **x** по ключу **k** с помощью функции	**tapply(x,k,FUN=…)**	53, **69**, 82
аналог **tapply** для фактора	**sapply(x,FUN=…)**	**34**
обобщить записи **x** по ключу **k** с помощью функции	**aggregate(x, by=list(k),FUN=…)**	**69**, **71**, 82
аналог **tapply** для фактора или списка	**by(data,INDICES,FUN=…)**	**69**
Обобщение данных		
сумма значений массива **x**	**sum(x)**	33, 68, 104
подсчет встречаемости значений **x**	**table(x)**	**35**, **44**, 68
возвращает вектор сглаженных относительных частот распределения значений исходного вектора	**density(x)**	**49**, 40, **49**
возвращает вектор накопленной суммы значений исходного вектора	**cumsum(x)**	
расчет плотности нормального распределения	**dnorm(…)**	**49**
обобщенная статистическая характеристика	**summary(x)**	81, 88, 90, 94, 97

возвращает вектор рангов всех значений исходного вектора	**rank(x)**	
возвращает среднюю арифметическую	**mean(x)**	10
возвращает стандартное отклонение	**sd(x)**	81-82
создает массив результатов частотного анализа значений исходного ряда	**hist(x, plot=FALSE)**	**46**
построение массива с частотами в интервалах	**cut(x,breaks)**	
Выявление особенных значений		
извлечение уникальных значений из массива **x**	**unique(x)**	**36**, 39
возвращает вектор с минимальным и максимальным значением исходного вектора	**range(x)**	
определение экстремальных значений вектора	**min(x), max(x)**	
извлекает **n** чисел из вектора **x** в случайном порядке; распределение равномерное	**sample(n,x)**	**26**, 79, 86
возвращает вектор центрированных (на среднюю), нормированных (на стандартное отклонение) значений	**scale(x)**	
Форматирование, вывод данных (оставляет **n** значащих цифр)	**print(x,n)**	**24-25**
разбивает диапазон значений исходного вектора на заданное число интервалов (**breaks =**) и возвращает массив значений, указывающий на принадлежность данной варианты данному интервалу	**cut(x, breaks =)**	
Статистический анализ		
расчет нормированных отклонений	**scale(x)**	
определить ранг, диапазон, минимум, максимум	**rank(x), range(x), min(x), max(x)**	
рассчитать среднюю, стандартное отклонение, медиану	**mean(x), sd(x), median(x)**	10, 81-82
дисперсионный анализ	**aov(y~x,p), glm(x)**	**88**, 94
корреляционный анализ всех переменных матрицы **m**	**cor(m)**	**91-92**
расчет линейной регрессии зависимости **y** от **x**	**lm(y~x)**	**93**, **94, 95**
расчет теоретических значений **y**	**predict(lm(y~x))**	**94**
настройка криволинейной модели	**nls(formula=,start=)**	**96**
расчет собственных векторов по ковариационной матрице	**eigen(var(x))**	**99**
расчет главных компонент по ковариационной матрице	**princomp(var(x))**	**99**
минимизации целевой функции **f**, перебором параметров **p**	**nlm(f,p), optim(p,f)**	**105**
Статистические тесты		
оценка значимости коэффициента корреляции	**cor.test(x,y)**	**91**
тест Шапиро – Уилка	**shapiro.test(x1,x2)**	**84**
тест Стьюдента	**t.test(x1,x2)**	**85**
тест Фишера	**var.test(x1,x2)**	**85**
тест Вилкоксона – Манна – Уитни	**wilcox.test(x1,x2)**	**84**

тест Тьюки (множествоенное сравнение)	**TukeyHSD(aov(y~x,p))**	**89**
тест Бартлетта (сравнение дисперсий)	**bartlett.test(y~x,p)**	**89**
тест Велша (дисперсионный анализ для выборок с неоднородными дисперсиями)	**oneway.test(y~x,p)**	**89**
тест Краскела – Уоллиса (непараметрический дисперсионный анализ)	**kruskal.test(y~x,p)**	**89**
Управление программой		
организация циклов выполнения операций	**for(i in диапазон){ }**	**39**, 86, **102-105**
создание пользовательской функции	**function(x,…){ return}**	**41**, **103-105**
создание условий выбора выполнения функций	**if(условие)функция else функция**	**87**
выделение из массива группы ячеек	**subset(x,cond)**	
Построение диаграмм		
создать поле для построения диаграммы и нарисовать диаграмму значений по оси ординат	**plot(x,y)**	14, 42, **45**, 46, 49, **52**, 69, 77, 79, 91, 93, 96, 99, 101, 103
создать диаграмму размаха по квартилям	**boxplot(x,…)**	14, 43, **47**, **53**, 83, 84, 88
создать диаграмму рассеяния точек	**stripchart (x,…)**	42, **47**
Создать столбчатую диаграмму	**barplot(x,…)**	**54**
добавить линию на созданную диаграмму	**lines(x,…)**	15, **48**, 48, 103
добавить точки на уже созданную диаграмму	**points(x,…)**	15, **48**, **56**, 79, 87, 93
добавить символы на диаграмму	**text(x,…)**	15, **48**, 55, 99
добавить легенду на диаграмму	**legend((позиция, список, аргументы)**	14, **50**, 52, 54, 87, 97
разметить и подписать оси диаграммы	**axis(x,…)**	14, **51**, 55
добавить график линии по коэффицентам	**abline(x,…)**	**87**
нарисовать гистограмму распределения частот	**hist(x,…)**	14, 26, 43, **45**, 86-87, 96
зарезервировать число полей для совместного отображения нескольких диаграмм	**par(n)**	
заново создать поле для диаграммы и нарисовать несколько графиков по оси ординат, используя разные столбцы двумерного массива	**matplot(x,…)**	**57**
создать двойной график по результатам компонентного анализа	**biplot(x,…)**	**99**

СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

Акиньшин, А. Функции в R [Электронный ресурс] / А. Акиншин. – Электрон. дан. – [2013]. – URL: https://aakinshin.net/ru/posts/r-functions/. – (дата обращения: 12.02.2021).

Введение в R – систему статистического анализа данных [Электронный ресурс]. – [2019]. – URL: http://mpoctok.narod.ru/r/intro.htm. – (дата обращения: 12.02.2021).

Ивантер, Э. В. Элементарная биометрия (с примерами на R) [Электронный ресурс] : учебное электронное пособие / Э. В. Ивантер, А. В. Коросов. – Электрон. дан. – Петрозаводск : Изд-во ПетрГУ, 2013. – URL: https://www.twirpx.org/file/1901647/. – (дата обращения: 12.02.2021).

Коросов, А. В. Компьютерная обработка биологических данных [Электронный ресурс] : учебное электронное пособие / А. В. Коросов, В. В. Горбач. – Электрон. дан. – Петрозаводск : Изд-во ПетрГУ, 2017. – URL: https://www.twirpx.org/file/2501217/ – (дата обращения: 12.02.2021), URL: https://elibrary.karelia.ru/book.shtml?id=29099#t20c. – (дата обращения: 25.02.2021).

Кушниренко, А. Г. Программирование для математиков : учебное пособие / А. Г. Кушниренко, Г. В. Лебедев. – Москва : Наука, 1988. – 384 с. – URL: https://www.niisi.ru/kumir/books/3.pdf. – (дата обращения: 14.02.2021).

Литтл, Дж. Статистический анализ данных с пропусками / Дж. Литтл, Л. Рубин. – Москва : Финансы и статистика, 1990. – 336 с. – URL: https://www.twirpx.org/file/249002/. – (дата обращения: 12.02.2021).

Мастицкий, С. Э. R: Анализ и визуализация данных [Электронный ресурс] / С. Э. Мастицкий. – Электрон. дан. – [2011]. – URL: https://r-analytics.blogspot.com/2011/04/blog-post.html. – (дата обращения: 12.02.2021).

Мастицкий, С. Э. Статистический анализ и визуализация данных с помощью R / С. Э. Мастицкий, В. К. Шитиков. – Москва : ДМК Пресс, 2015. 496 с. – URL: http://www.ievbras.ru/ecostat/Kiril/R/MS_2014/MS_2014.pdf. – (дата обращения: 12.02.2021).

Уолш, Б. Программирование на Бейсике / Б. Уолш. – Москва : Радио, 1988. – 336 с. – URL: https://www.twirpx.org/file/1607718/. – (дата обращения: 12.02.2021).

Шитиков, В. К. Классификация, регрессия и другие алгоритмы Data Mining с использованием R [Электронный ресурс] / В. К. Шитиков, С. Э. Мастицкий. – Электрон. дан. – [2017]. – URL: https://www.twirpx.org/file/2203014/, https://ranalytics.github.io/data-mining/. – (дата обращения: 12.02.2021).

Шитиков, В. К. Рандомизация и бутстреп: статистический анализ в биологии и экологии с использованием R / В. К. Шитиков, Г. С. Розенберг. – Тольятти : Кас-

сандра, 2013. – 314 с. – URL: http://www.ievbras.ru/download/Random.pdf. – (дата обращения: 12.02.2021).

Kabacoff, R. I. Quivk-R [Электронный ресурс] / R. I. Kabacoff. – Электрон. дан. – [2017]. – URL: https://www.statmethods.net/. – (дата обращения: 12.02.2021).

СОДЕРЖАНИЕ

Учебное электронное издание

Коросов Андрей Викторович
Горбач Вячеслав Васильевич

ПРАКТИЧЕСКОЕ ВВЕДЕНИЕ В СРЕДУ R

Учебное пособие для обучающихся по направлениям подготовки «Биология» и «Экология и природопользование»

Редактор *О. В. Обарчук*

Оригинал-макет, электронная версия и оформление обложки

А. В. Коросов, В. В. Горбач

Подписано к изготовлению 15.06.2021. 1 CD-R. 1.5 Мб.

Тираж 100 экз. Изд. № 46

Федеральное государственное бюджетное образовательное учреждение высшего образования
ПЕТРОЗАВОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
185910, г. Петрозаводск, пр. Ленина, 33

https://petrsu.ru
Тел.: (8142) 71-10-01

Изготовлено в Издательстве ПетрГУ
185910, г. Петрозаводск, пр. Ленина, 33
URL: press.petrsu.ru/UNIPRESS/UNIPRESS.html
Тел./факс: (8142) 78-15-40
nvpahomova@yandex.ru