Глава 2

Данные и графики

Анализ «хороших» данных — это просто. А вот чтобы сделать Ваши данные «хорошими», а затем и представить их — придётся попотеть.

2.1. R и работа с данными

Подготовка данных к работе — это одна из самых больших проблем для новичка в **R**. Сама по себе обработка данных подробно описана в разных руководствах и пособиях, а вот информация как добиться того, чтобы **R** прочитал приготовленные в другой программе данные, как правило, опускается. Почему это так очевидно: входные данные могут иметь слишком разный формат, чтобы написать по этому вопросу исчерпывающее и компактное руководство.

Данные можно представить в текстовом или в бинарном виде. Не вдаваясь в детали, примем, что текстовые данные — это данные, которые можно прочитать и отредактировать в текстовом редакторе (Emacs/Vi и прочее). Для того, чтобы отредактировать бинарные данные, как правило, нужна программа, которая эти данные произвела. Текстовые данные для статистической обработки — это текстовые таблицы, где каждая строка соответствует строчке таблицы, а колонки определяются при помощи разделителей. Обычно в качестве разделителей текстовых данных используются пробельные символы (пробел, табуляция и тому подобное), запятые или точки с запятой.

Первое что надо сделать перед чтением данных — это убедиться, что текущая директория в **R** и та директория, где находятся данные одно и то же. Для этого в запущенной сессии **R** надо ввести команду:

```
> getwd()
[1] "/home/username/"
```

Пусть это вовсе не та директория, в которой лежат данные. Поменять рабочую директорию можно командой:

```
> setwd("./workdir")
> getwd()
[1] "/home/username/workdir"
```

Как обычно, развёрнутую справку можно получить с помощью функции вызова справки `help(getwd)`. Далее следует проверить, а есть ли в текущей директории нужный файл:

```
> dir()
[1] "mydata.txt"
```

Вот теперь можно и загрузить данные. За чтение табличных текстовых данных отвечает команда `read.table()`:

```
> read.table("mydata.txt", sep=";", head=TRUE)
  a б в
1 1 2 3
2 4 5 6
3 7 8 9
```

Всё очень просто за исключением того, что перед чтением нужно знать в каком формате хранятся данные. То есть то, что у столбцов есть имена (`head=TRUE`) и разделителем является точка с запятой (`sep=";"`). Функция `read.table()` очень хороша, но не настолько умна, чтобы определять формат данных на лету. Чтобы посмотреть содержимое файла не выходя из **R**, можно воспользоваться функцией `file.show()`:

```
> file.show("mydata.txt")
a;б;в
1;2;3
4;5;6
7;8;9
```

В **R** многие команды, в том числе и `read.table()`, имеют для аргументов значения по умолчанию. Например, значение `sep` по умолчанию равно `""`. В данном случае это означает, что разделителем является любое количество пробелов или знаков табуляции, поэтому если данные вместо точек с запятыми разделены пробельными символами, то аргумент `sep` можно не указывать. Естественно, бывает безумное множество различных частных случаев, и сколько бы усилий не было приложено, всё не описать. Отметим, однако, ещё несколько важных моментов:

1) Файлы можно загружать и из других директорий, при этом можно использовать относительную адресацию:

```
> read.table("../workdir/mydata.txt")
```

2) Русский текст в файлах читается без проблем, если он набран в кодировке совпадающей с текущей локалью. Пусть локаль `ru_RU.UTF-8`, а сам файл закодирован в KOI8-R, тогда при его чтении следует воспользоваться функцией `file()`:

```
> read.table(
+  file("mydata-unicode.txt", encoding="KOI8-R"),
+                                sep=";", head=TRUE)
  а б в
1 1 2 3
2 4 5 6
3 7 8 9
```

3) Иногда нужно, чтобы **R** прочитал кроме имён столбцов ещё и имена строк. В этом случае в первой строке должно быть на одну колонку меньше, чем в теле таблицы (в данном примере три вместо четырёх):

```
> file.show("mydata2.txt")
а б в
раз 1 2 3
два 4 5 6
три 7 8 9
> read.table("mydata2.txt", head=TRUE)
    а б в
раз 1 2 3
два 4 5 6
три 7 8 9
```

4) По отечественным правилам в качестве десятичного разделителя нужно использовать запятую, а не точку. Если кто-то при подготовке исходных данных этим правилам последовал, то необходимо переопределить аргумент `dec`:

```
> read.table("mydata3.txt", dec=",", h=T)
      а   б   в
раз 1.1 2.2 3.3
два 4.4 5.0 6.0
три 7.0 8.0 9.0
```

Обратите внимание на сокращённое обозначение аргумента и его значения (`h=T`). Сокращать можно и нужно, но с осторожностью, поэтому в далее в этом тексте всегда будет именно `TRUE/FALSE`.

В целом, с текстовыми таблицами больших проблем не возникает. Разные экзотические текстовые форматы, как правило, можно преобразовать к «типичным» если не с помощью **R**, то с помощью каких-нибудь многочисленнейших текстовых утилит (вплоть до «тяжеловесов» типа языка Perl). А вот с «посторонними» бинарными форматами дело обстоит гораздо хуже. Здесь, прежде всего, возникают проблемы, связанные с полностью закрытыми форматами, например, такими как формат популярной в определённых кругах программы MS Excel. Вообще говоря, ответ на вопрос: «Как прочитать бинарный формат в **R**?» — часто сводится к совету по образцу известного анекдота: «выключим газ, выльем воду и вернёмся к условию предыдущей задачи». То есть надо найти способ, который позволит преобразовать бинарные данные в обычные текстовые таблицы. Проблем на этом пути возникает обычно не слишком много, но уж больно они разнообразные.

Второй путь — это найти способ прочитать данные в **R** без преобразования. Специально для этих целей в **R** есть пакет **foreign**, который может читать бинарные данные, выводимые пакетами Minitab, S, SAS, SPSS, Stata, Systat, а также формат DBF. Чтобы узнать подробнее об определённых в этом пакете командах, надо загрузить пакет и вызвать общую справку:

```
> library(foreign)
> help(package=foreign)
```

Что же касается всё того же пресловутого формата Excel, то здесь дело хуже. Есть не меньше пяти разных способов, как загружать в **R** эти файлы, но все они имеют ограничения. К тому же новый формат MS Excel 2007 пока вообще не поддерживается. Из всех способов наиболее привлекательным представляется обмен с **R** через буфер. Если открыть в OpenOffice Calc xls-файл , то можно скопировать в буфер любое количество ячеек, а потом загрузить их в **R**:

```
> read.table("clipboard")
```

Это очень просто, и главное, работает с любой Excel-подобной программой.

Тут следует отметить ещё одну вещь: *ни в коем случае не рекомендуется производить какой-либо статистический анализ в программах электронных таблиц.* Не говоря уже о том, что интернет просто забит статьями об ошибках в этих программах и/или в их статистических модулях, это ещё и крайне неверно идеологически. Иначе говоря:

Используйте R!

Добавим ещё несколько деталей:

1) **R** может загружать изображения. Для этого есть несколько пакетов. Наиболее продвинутый из них — это пакет **pixmap**. **R** также может загружать карты в формате ArcInfo и др. (пакеты **maps**, **maptools**) и вообще много чего ещё.

2) У **R** есть собственный бинарный формат. Он быстро записывается и быстро загружается, но его нельзя использовать для передачи данных.

```
> x <- "яблоко"
> save(x, file="x.rd")
> rm(x)
> x
Ошибка: объект "x" не найден
> dir()
[1] "x.rd"
> load("x.rd")
> x
[1] "яблоко"
```

Для сохранения и загрузки бинарных файлов служат команды `save()` и `load()`, для создания объекта — `<-`, а для удаления — `rm()`.

3) Для **R** написано множество интерфейсов к базам данных, в частности, для MySQL, PostgresSQL и SQLite (последний может вызываться прямо из **R**, см. пакеты **RSQLite** и **sqldf**).

4) Наконец, **R** сам может записывать таблицы и другие результаты обработки данных, и, разумеется, графики. Об этом мы поговорим ниже.

2.2. Графики

Несмотря на то, что «настоящие» статистики часто относятся к графикам почти с презрением, для «широких масс» одним из основных достоинств **R** служит именно удивительное разнообразие типов графиков, которые он может построить. **R** в этом смысле — один из рекордсменов. В базовом наборе есть несколько десятков типов графиков, ещё больше в рекомендуемом пакете **lattice**, и, естественно, намного больше в пакетах с CRAN. По оценочным прикидкам получается, что разнообразных типов графиков в **R** никак не меньше тысячи. При этом они все ещё достаточно хорошо настраиваются, то есть пользователь при желании достаточно легко может разнообразить эту исходную тысячу на свой вкус.

2.2.1. Два типа графических команд

Для правильного отображения кириллицы в X-окне (если это действительно необходимо) для начала следует правильно указать шрифты, например так:

```
> X11(fonts = c(
+   "-rfx-helvetica-%s-%s-*-*-%d-*-*-*-*-*-koi8-r",
```

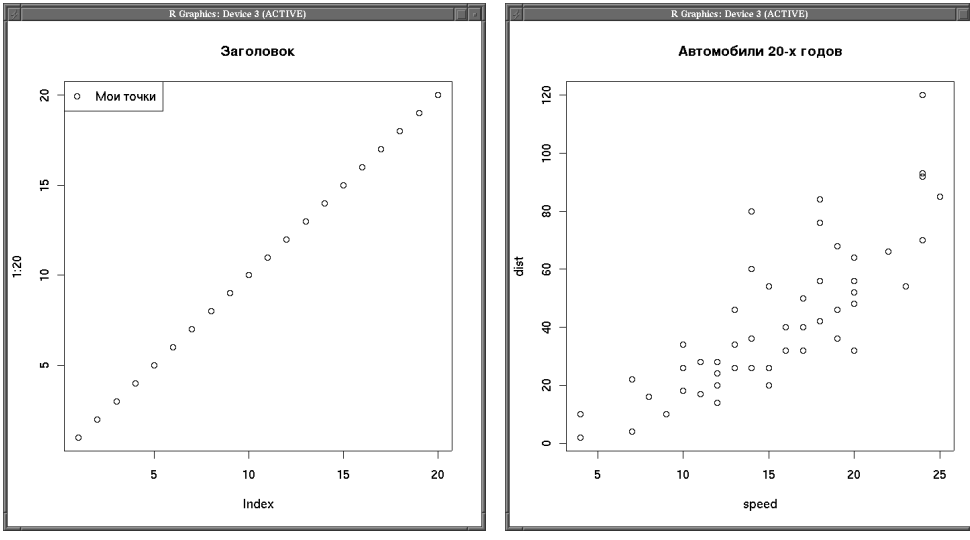

Рис. 2.1. Окна X11 с простейшим графиком

```
+      "-adobe-symbol-medium-r-*-*-%d-*-*-*-*-*-*-*-*"))
```

Опция `fonts` команды `X11` принимает вектор из двух элементов, который формируется с помощью команды `c()`. Первый элемент соответствует шрифту, используемому для обычных текстовых меток, а второй для отображения стандартных спецсимволов, например, греческого алфавита. С помощью программы **xfontsel** можно подобрать себе шрифт по вкусу. Подробности в разделе Fonts в документации, выдаваемой по `help(X11)`.

Рассмотрим пример:

```
> plot(1:20, main="Заголовок")
> legend("topleft", pch=1, legend="Мои␣точки")
```

Тут много такого, о чём пока речи не шло. Но самое главное это то, что первая команда рисует график «с нуля», тогда как вторая только добавляет к уже нарисованному графику детали. Это и есть два типа графических команд, используемых в базовом графическом наборе **R**. Теперь немного подробнее: `plot()` — основная графическая команда, причём команда «умная» (правильнее сказать «generic», или общая). Это значит, что она распознаёт тип объекта, который подлежит рисованию, и строит соответствующий график. Например, в приведённом примере `1:20` — это последовательность чисел от 1 до 20, то есть вектор, а для одиночного вектора предусмотрен график, где по оси абсцисс — индексы (номера каждого элемента вектора по порядку), а по оси ординат — сами эти элементы.

Если в аргументе команды будет что-то другое, то будет построен иной, более подходящий для этого объекта, график. Вот пример:

```
> plot(cars)
> title(main="Автомобили␣20-х␣годов")
```

Здесь тоже команды обоих типов, оформленные немного иначе. Не беда, что мы забыли дать заголовок в команде `plot()`, так как его всегда можно добавить потом, командой `title()`. «cars»[1] — это встроенная в **R** таблица данных, которая использована здесь по прямому назначению, то есть для демонстрации возможностей программы. Для нас сейчас важно, что это — не вектор, а таблица из двух колонок: speed и distance (скорость и тормозная дистанция). Функция `plot()` автоматически нарисовала, так называемый, scatterplot, когда по оси X откладывается значение одной переменной (колонки), а по оси Y — другой, и ещё присвоила осям имена этих колонок. Любопытным советуем проверить, что нарисует `plot()`, если ему «подложить» таблицу с тремя колонками, скажем, встроенную таблицу «trees». Кстати говоря, узнать, какие ещё есть встроенные таблицы, можно с помощью команды `data()` (именно так, без аргументов).

2.2.2. Графические устройства

Когда вводится команда `plot()`, **R** открывает, так называемое, экранное графическое устройство[2] и начинает вывод на него. Если следующая команда того же типа, то есть не добавляющая, то **R** «сотрёт» старое изображение и начнёт выводить новое в этом же окне. Если ввести команду:

```
> dev.off()
```

то **R** закроет графическое окно, что, впрочем, можно сделать, просто щёлкнув по кнопке закрытия окна. Экранных устройств в **R** предусмотрено несколько, в каждой операционной системе своё (а в Mac OS X даже два). Но всё это не так важно, пока не захочется строить графики и сохранять их в виде графических файлов. В этом случае придётся познакомиться с другими графическими устройствами. Их несколько (количество опять-таки зависит от операционной системы), а пакеты предоставляют ещё около десятка. Работают они примерно так:

```
> png(file="1-20.png", bg="transparent")
> plot(1:20)
> dev.off()
```

Команда `png()` открывает одноимённое графическое устройство, причём задаётся параметр, включающий прозрачность базового фона (удобно, например,

[1] Прочитать, что такое «cars», можно, вызвав справку обычным образом ("?cars").
[2] В случае использования X Window — это стандартное окно X11.

для Web). Такого параметра у экранных устройств нет. Как только вводится команда `dev.off()`, устройство закрывается и на диске появляется файл `1-20.png`. `png()` — одно из самых распространённых устройств при записи файлов. Недостатком его является, разумеется, растровая природа этого формата. Аналогичным по своей функциональности является и устройство `jpeg()`, которое производит jpeg-файлы.

R поддерживает и векторные форматы, например, PDF. Здесь, однако, могут возникнуть специфические для русскоязычного пользователя трудности со шрифтами. Остановимся на этом чуть подробнее. Вот как надо «правильно» создавать PDF-файл, содержащий русский текст:

```
> pdf("1-20.pdf",family="NimbusSan",encoding="KOI8-R.enc")
> plot(1:20, main="Заголовок")
> dev.off()
> embedFonts("1-20.pdf")
```

Как видим, требуется указать, какой шрифт мы будем использовать, а также кодировку, с которой работаем. Другие доступные однобайтные кириллические кодировки, идущие с **R**: `CP1251.enc` и `KOI8-U.enc`. Затем нужно закрыть графическое устройство и *встроить в полученный файл шрифты* с помощью команды `embedFonts()`. Следует отметить, что шрифт NimbusSan и возможность встраивания шрифтов командой обеспечивается взаимодействием **R** со сторонней программой Ghostscript, в поставку которой входят шрифты, содержащие русские буквы. Кроме PDF, **R** «знает» и другие векторные форматы, например, PostScript, xfig и picTeX. Есть отдельный пакет **RSvgDevice**, который поддерживает популярный векторный формат SVG. График в этом формате можно, например, открыть и видоизменить в свободном векторном редакторе Inkscape.

2.2.3. Графические опции

Как уже говорилось, графика в **R** настраивается в очень широких пределах. Один из способов настройки — это видоизменение графических опций, встроенных в **R**. Вот, к примеру, распространённая задача: нарисовать две гистограммы одну под другой на одном рисунке. Чтобы это сделать, надо изменить исходные опции, а именно разделить пространство рисунка на две части, примерно так:

```
> # Создаётся eps-файл размером 6 на 6 дюймов
> postscript("2hist.eps",width=6.0,height=6.0,
+    horizontal=FALSE,onefile=FALSE,paper="special")
> # Изменяется одно из значений по умолчанию
> old.par <- par(mfrow=c(2,1))
> hist(cars$speed)
> hist(cars$dist)
> # Восстанавливаем старое значение по умолчанию
```

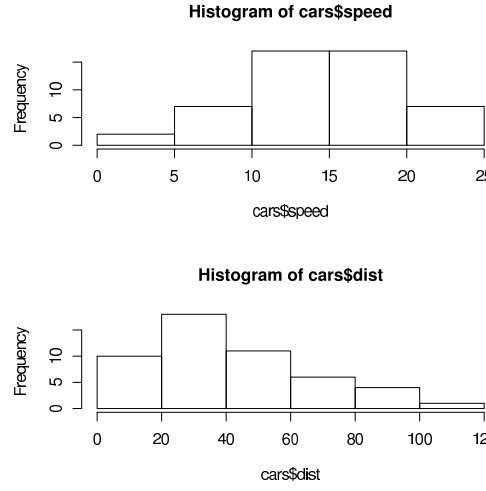

Рис. 2.2. Две гистограммы друг под другом

```
> par(old.par)
> dev.off()
```

Ключевая команда здесь `par()` — изменяется один из её параметров, `mfrow`, который регулирует сколько изображений и как будет размещено на «листе». Значение `mfrow` по умолчанию — `c(1,1)`, то есть один график по вертикали и один по горизонтали. Чтобы не печатать каждый раз команду `par()` без аргументов (для того чтобы выяснить умалчиваемые значения каждого из 71 параметра), мы «запомнили» старое значение в объекте `old.par`, а в конце вернули состояние к запомненному. То, что команда `hist()` строит гистограмму, очевидно из названия.

2.2.4. Идеологически верная графика

Несмотря на своё разнообразие, графическая система в **R** построена на основе строгих правил. Выбор типа графика, основных цветов и символов для изображения точек, расположение подписей и т. д. был тщательно продумано создателями. Одним из ключевых для **R** исследований является книга Уильяма Кливленда «Элементы графической обработки данных». Многие его идеи были осуществлены именно в S-PLUS, а затем и в **R**. Например, Кливленд нашёл, что традиционные «столбчатые» графики очень плохо воспринимаются, особенно когда речь идёт о близких значениях данных, и предложил им на замену: «точечные диаграммы». Вот так они реализованы в **R**:

```
> dotchart(Titanic[,,"Adult","No"],
```

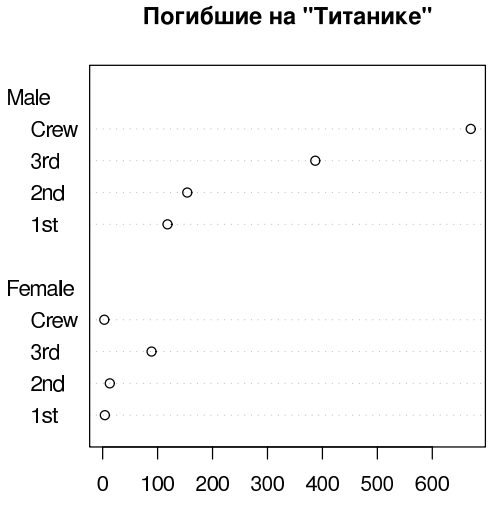

Рис. 2.3. Точечная диаграмма

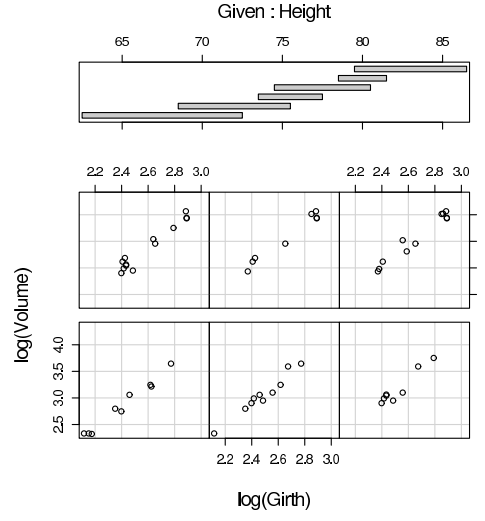

Рис. 2.4. График-решётка

```
+                    main='Погибшие на "Титанике"')
```

Встроенная таблица данных Titanic — это четырёхмерная матрица, которая отражает статистику по возрастным группам, типу билета и полу.

Особенно активно Кливленд (и далеко не только он) возражал против использования трёхмерных графиков и так называемых «пирогов». Поначалу «пирожных» графиков в **R** вовсе не было[3], причём по принципиальным соображениям. Трёхмерных графиков в **R** и сейчас немного (правда, есть особый пакет **rgl**, который позволяет строить такие графики на базе OpenGL), а если Вы хотите узнать, как меняется поведение двух переменных по отношению к третьей, **R** предлагает так называемые «Trellis graphs» или графики-решётки:

```
> coplot(log(Volume) ~ log(Girth) | Height, data = trees)
```

При выполнении этой команды на рисунке отображается как зависит объём древесины от объёма кроны (в логарифмической шкале) у деревьев различной высоты. Действительно, такое представление гораздо эффективнее трёхмерного. Странно, что распространённые пакеты статобработки почти не используют графики-решётки, хотя их наличие неоднократно называлось одной из главных причин коммерческого успеха S-PLUS.

[3]Сейчас они есть, но если Вы откроете страницу помощи, то узнаете, что «Pie charts are a very bad way of displaying information».

2.2.5. Интерактивность

Интерактивная графика позволяет выяснить, где именно на графике расположены нужные Вам точки и разместить объект (скажем, подпись) в нужное место, а также проследить «судьбу» одних и тех же точек на разных графиках. Кроме того, если данные многомерные, то можно вращать облако точек в плоскости разных переменных с тем чтобы выяснить структуру данных.

Ещё несколько лет назад пришлось бы написать, что здесь вместо **R** следует воспользоваться другими аналитическими инструментами, но **R** развивается так быстро, что все эти методы теперь доступны, причём в нескольких вариантах. Например, так можно добавлять подписи в указанную мышкой область графика:

```
> plot(1:20)
> text(locator(), "Моя␣любимая␣точка", pos=4)
```

После того как введена вторая команда, надо щёлкнуть левой кнопкой мыши на выбранной точке в графике, а затем уже без разницы где щёлкнуть правой кнопкой мыши.

Интерактивная графика других типов реализована командой `identify()`, а также пакетами **rggobi**, **TeachingDemos** и **iplot**.

2.3. Как сохранять результаты

Начинающие работу с **R** обычно просто копируют результаты работы (скажем, данные тестов) из консоли **R** в текстовый файл. И действительно, на первых порах этого может оказаться достаточно. Однако рано или поздно возникает необходимость сохранять объёмные объекты (например, таблицы данных), созданные в течении работы. Можно использовать уже упомянутый в начале статьи внутренний бинарный формат, но это не всегда удобно. Лучше всего сохранять таблицы данных в виде текстовых таблиц, которые потом можно будет открывать другими приложениями или текстовыми редакторами. Для этого служит команда `write.table()`:

```
> write.table(file="trees.csv", trees,
+             row.names=F, sep=";", quote=F)
```

В текущую рабочую директорию будет записан файл `trees.csv`, образованный из встроенной в **R** таблицы данных trees. А что, если надо записать во внешний файл результаты выполнения команд? В этом случае используется команда `sink()`:

```
> sink("1.txt", split=T)
> 2+2
[1] 4
> sink()
```

В этом случае во внешний файл запишется строчка «[1] 4», то есть результат выполнения команды. Сама команда записана не будет, а если хочется, чтобы она была записана, то придётся ввести что-то вроде:

```
> print("2+2")
[1] "2+2"
> 2+2
[1] 4
```

то есть повторять каждую команду два раза. Для сохранения истории команд служит команда `savehistory()`, а для сохранения всех созданных объектов — `save.image()`. Последняя может оказаться также полезной для сохранения промежуточных результатов работы, если не уверены в стабильности работы компьютера.

2.4. Мастера отчётов

Таблицы, созданные в **R**, можно сохранять и в более «приличном» виде, например, в форматах LaTeX ([1]) или HTML, при помощи пакета **xtable**. Естественно, хочется пойти дальше, и сохранять в каком-нибудь из этих форматов вообще всю **R** -сессию. Для HTML такое возможно, если использовать пакет **R2HTML** с CRAN:

```
> library(R2HTML)
> dir.create("example")
> HTMLStart("example")
HTML> 2+2
HTML> plot(1:20)
HTML> HTMLplot()
HTML> HTMLStop()
>
```

В рабочей директории будет создана поддиректория `example` и туда будут записаны HTML-файлы, содержащие полный отчёт о текущей сессии, в том числе и созданный график.

Можно пойти и ещё дальше. Что, если создать файл, который будет содержать код **R**, перемешанный с текстовыми комментариями, и потом «скормить» этот файл **R** так, чтобы фрагменты кода заменились на результат их исполнения? Идея эта называется «literate programming» (грамотное программирование) и принадлежит Дональду Кнуту, создателю TeX. В случае **R** такая система используется для автоматической генерации отчётов — «фичи», которая фактически отсутствует в остальных статистических пакетах и делает **R** поистине незаменимым. Для создания подобного отчёта, для начала, надо набрать простой файл с LaTeX-подобной структурой и назвать его, например, `test-Sweave.Rnw`:

```
\documentclass[a4paper,12pt]{scrartcl}
% Стандартная шапка для \LaTeX-документа
\usepackage[T2A]{fontenc}
% В зависимости от используемой локали вместо utf8 нужно
%поставить cp1251 или koi8-r
\usepackage[utf8]{inputenc}
\usepackage[english,russian]{babel}
\usepackage{indentfirst}

\title{Тест Sweave}
\author{A.B.\,Top}
\begin{document} % Тело документа
\maketitle

\textsf{R} как калькулятор:
<<echo=TRUE,print=TRUE>>=
1 + 1
1 + pi
sin(pi/2)
@

Картинка:
<<fig=TRUE>>=
plot(1:20)
@

\end{document}
```

Затем этот файл необходимо обработать в **R**:

```
> Sweave("test-Sweave.Rnw")
Writing to file test-Sweave.tex
Processing code chunks ...
 1 : echo print term verbatim
 2 : echo term verbatim eps pdf

You can now run LaTeX on 'test-Sweave.tex'
```

При этом создаётся готовый LaTeX-файл `test-Sweave.tex`. И, наконец, при помощи **latex**/**dvips** или **pdflatex** получить результирующий файл:

```
=> latex test-Sweave.tex
=> dvips test-Sweave.dvi
=> gv test-Sweave.ps
```

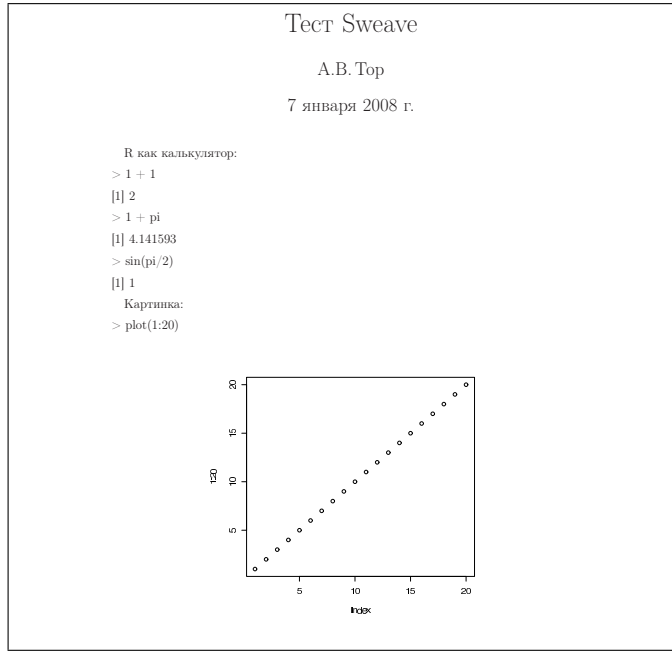

Рис. 2.5. Пример отчёта полученного с помощью команды Sweave

```
# или
=> pdflatex test-Sweave.tex
=> acroread test-Sweave.pdf
```

Такой отчёт можно расширять, шлифовать, изменять исходные данные, и при этом усилия по оформлению практически сводятся на нет. Если есть желание, чтобы код **R** набирался моноширинным шрифтом, то в LaTeX-преамбуле Rnw-файла следует добавить строчку:

```
\usepackage[noae]{Sweave}
```

Исходный код и авторскую документацию профессора Фридриха Лайша (Friedrich Leisch) можно найти здесь: `http://www.ci.tuwien.ac.at/~leisch/Sweave/`.

Есть и другие системы генерации отчётов, например, уже упомянутый пакет **R2HTML** умеет производить похожие отчёты в HTML. Есть пакет **brew**, который позволяет создавать автоматические отчёты в текстовой форме (разумеется, без графиков), и совсем новый пакет **odfWeave**, который может работать с ODF (формат OpenOffice.org).